Principled Design and Runtime Analysis of Abstract Convex Evolutionary Search

Alberto Moraglio a.moraglio@exeter.ac.uk
Department of Computer Science, University of Exeter, Exeter EX4 4QF, UK

Dirk Sudholt d.sudholt@sheffield.ac.uk
Department of Computer Science, University of Sheffield, Sheffield S1 4DP, UK

doi:10.1162/EVCO_a_00169

Abstract

Geometric crossover is a formal class of crossovers that includes many well-known recombination operators across representations. In previous work, it was shown that all evolutionary algorithms with geometric crossover (but 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 toward linking a geometric theory of representations and runtime analysis in the attempt to (1) set the basis for a more general, unified approach for the runtime analysis of evolutionary algorithms across representations, and (2) 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.

Keywords

Geometric crossover, recombination, runtime analysis, convex search, pure adaptive search.

1 Introduction

The evolutionary computation (EC) field needs unification and systematization in a rational framework to survive its own success (De Jong, 2006).

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.

A preliminary version of this paper with parts of the results was published as Moraglio and Sudholt (2012). This extended version contains new results about the optimal choice of the population size in convex evolutionary search on binary spaces.

Manuscript received: April 20, 2015; revised: September 28, 2015; accepted: October 2, 2015.

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 possible? Would such a general framework be able to capture essential properties encompassing all EAs, or would it be too abstract to say anything useful? These are important, difficult open research questions that the present work explores.

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 (1991) pioneered a unified theory of representations; Langdon and Poli (2002) unified schema theories for traditional genetic algorithms and genetic programming; Stephens and Zamora (2003) suggested that all evolutionary algorithms can be unified using dynamic systems and coarse graining; Rothlauf (2002) initiated a popular but less formal theory of representations; Rowe et al. (2002), building upon Radcliffe's work, devised a theory of representations based on group theory; Reidys and Stadler (2002) built a theory of fitness landscapes that connects with representations and search operators; Mitavskiy (2004) devised category-theoretic methods to compare rigorously evolutionary algorithms with different representations.

For the no free lunch (NFL) theorem (Wolpert and Macready, 1996), 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 with geometric crossover introduced by Moraglio and Poli (2004) encompasses and formalises many well-known representation-specific EAs (Moraglio, 2007). Moraglio (2011) 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 that 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 (Moraglio, 2011).

Runtime analysis is the standard approach to analytically analyse algorithmic performance. It has been applied with an ever increasing success to randomised search heuristics, and it is establishing itself as a leading theory (Neumann and Witt, 2010; Auger and Doerr, 2011; Jansen, 2013). 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 (Jansen and Wegener, 2002; 2005; Kötzing et al., 2011). It was demonstrated how crossover speeds up search by combining features, or building blocks, of good solutions (Sudholt, 2012; Doerr et al., 2015). Furthermore, crossover proved beneficial for combinatorial problems, including colouring problems (Sudholt, 2005) and the all-pairs shortest path problem (Doerr et al., 2012).

Analyses like these are done on a per-algorithm, 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: He and Yao, 2004; Doerr et al., 2010; Lehre and Witt, 2014, and fitness levels: Sudholt, 2013; Corus et al., 2014). However, their applications are specific for an algorithm-problem pair and require 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 represent 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 toward 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), originally introduced by Patel et al. (1988), which is an *ideal* algorithm that has been thoroughly studied 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 results from Moraglio (2011).[1] 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 of representation-specific language, that is, binary strings, bits, and schemata.

[1]The framework in this section may seem related to the field of convex optimisation, but it differs fundamentally from it. See Moraglio (2011) for a discussion of the differences.

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 $d : X \times X \longrightarrow \mathbf{R}$ 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 $x, y, z \in X$:

- $d(x, y) \geq 0$ and $d(x, y) = 0$ iff $x = y$.
- $d(x, y) = d(y, x)$.
- $d(x, z) + d(z, y) \geq d(x, y)$.

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 $[x, y]_d = \{z \in X | d(x, z) + d(z, y) = d(x, y)\}$. 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* (van de Vel (1993)) $\mathcal{C}$ 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, namely, 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, that is, $co(A) = \bigcap \{C | A \subseteq C \in \mathcal{X}\}$.

In the Hamming space, the notions of segment, convex set, and schema coincide,[2] as illustrated in the following. Let $H(a, b)$ be the schema obtained from the binary strings a and b by positionwise 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

[2]The 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.

the product of the convexities on each dimension (i.e., product convexity) induced by the trivial metric (van de Vel (1993)). Consequently, the intersection of two schemata is a schema or the empty set (e.g., $\texttt{**101} \cap \texttt{1**01} = \texttt{1*101}$) 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., $co(0101, 1001, 0000) = \texttt{**0*}$).

2.2 Formal Evolutionary Algorithm and Abstract Convex Evolutionary Search

DEFINITION 1 ((GEOMETRIC CROSSOVER) (Moraglio and Poli, 2004)): *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 that is defined on the 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 with respect to the Hamming distance. The class of geometric crossover is really broad and extends far beyond this simple example, as shown by Moraglio (2007). 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, subtree 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

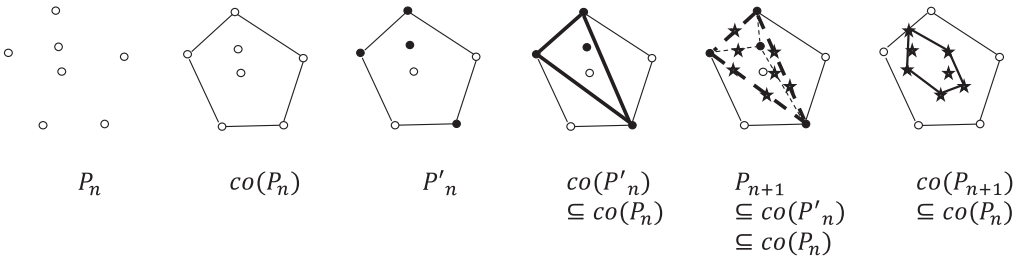

Figure 1: Convex evolutionary search.

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 Surry and Radcliffe (1996).

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 behaviour 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 behaviour, that is, those behavioural properties common to all specific behaviours obtained by assigning all possible specific values to the parameter left unspecified. We term *abstract evolutionary search* the behaviour of a formal evolutionary algorithm in which the underlying metric d is unspecified. As this behaviour is inferred from the formal evolutionary algorithm and the metric axioms only, *it is the behaviour of the formal evolutionary algorithm on all possible search spaces and associated representations.*

The abstract behaviour of a formal evolutionary algorithm is a formal object itself based on the metric axioms. In the following sections, we show that the behaviour 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 (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 maximising (see Figure 2).[3] This is because the

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

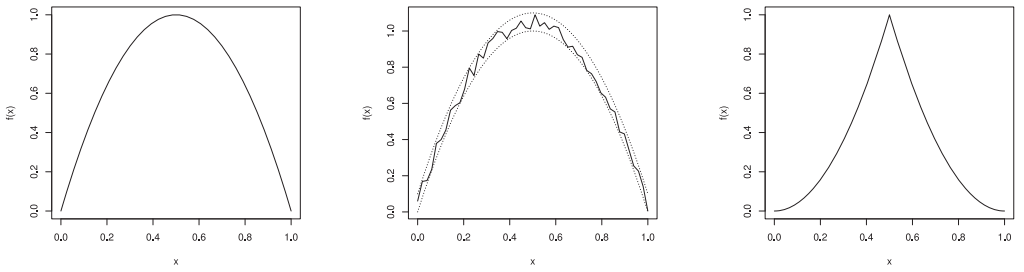

Figure 2: Examples of concave function (*left*), ϵ-approximately concave function (*middle*), and quasi-concave function that is not concave (*right*) on the real line.

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 by Moraglio (2011):

- *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 $x, y : z \in [x, y]$, $f(z) \geq \min(f(x), f(y))$.

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 OneMax 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 these 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.

Moraglio (2011) showed 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 rest of this work, we investigate the possibility of a general runtime analysis of evolutionary algorithms on concave landscapes *across representations* starting from the preceding result. 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 (see Section 3).

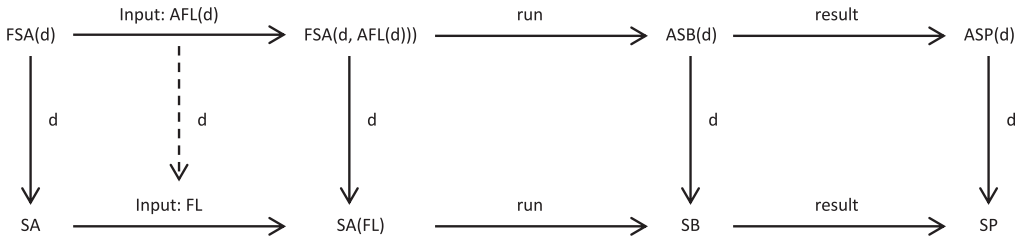

Figure 3: Functional relations between search algorithm, fitness landscape, search behaviour, search performance, and their abstract counterparts.

Algorithm 1 Convex search algorithm

```
Input: μ: population size
Output: individual in the last population

Initialise population uniformly at random
while population has not converged to the same individual do
    Rank individuals on fitness
    if there are at least two fitness values in the current population then
        remove all individuals with the worst fitness
    end if
    Apply μ times metric convex hull uniform recombination to the remaining
    individuals to create the next population
end while
return any individual in the last population
```

2.4 Abstract-to-Specific Interface

Figure 3 illustrates the envisioned functional relation between search algorithm (SA), fitness landscape (FL), search behaviour (SB), and search performance (SP) and their abstract counterparts, formal search algorithm (FSA), abstract fitness landscape (AFL), abstract search behaviour (ABS), and abstract search performance (ASP). The horizontal arrows at the bottom mean that the algorithm SA is fed with the parameter fitness landscape FL, which when run together gives rise to the search behaviour SB that produces the search performance SP. The horizontal arrows at the top mirror those at the bottom and depict analogous relations at an abstract level in which the underlying distance d is left unspecified. The vertical arrows relate abstract and concrete levels by functional application of the functional forms at 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 allows 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),

Algorithm 2 Binary uniform convex hull recombination

```
for each position in the binary strings do
  if all parents have 1 or 0 at that position then
    the offspring has 1 or 0 at that position, respectively
  else
    (if there is at least one 1 and at least one 0 at that position)
    the offspring has 1 or 0 at that position with probability .5
  end if
end for
```


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; 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 multiparental recombination operator that 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 rewriting 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 recombination that 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 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 finite,[4] with values $a_0 < a_1 < \cdots < a_q$. 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:

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

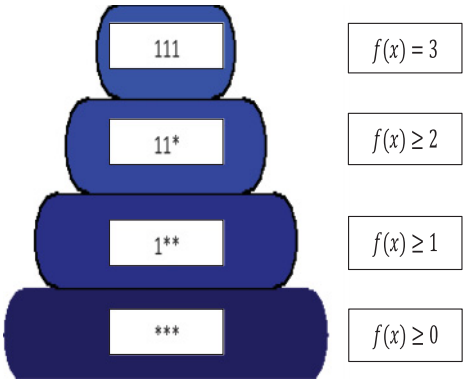

Figure 4: Level sets of LeadingOnes.

$L_0 \supsetneq L_1 \supsetneq \cdots \supsetneq L_q$. In particular, $L_0 = S$. The sets $(L_0 \setminus L_1), (L_1 \setminus L_2), \ldots, L_q$ form a *fitness-level partition* in the sense defined by Wegener (2002).

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

- If f is quasi-concave: for all sets $C \subseteq S$, if $z \in \text{co}(C)$, then $f(z) \geq \min\{f(x) \mid x \in C\}$.
- 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 the 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, namely, $f(x_1) := 1$. 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 characterised as follows.[5] Let $w_1, \ldots, w_n$ 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 set correctly with respect to the target t whose positions are specified by the first k entries of p has fitness $w_1 + \cdots + w_k$.

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 the following:

- The number $q + 1$ of fitness levels is polynomial in n (problem size, $n = \log(|S|)$).
- 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 the 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.

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

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 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, that is, 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 that 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 into 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 1980s 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 with 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 covered in a monograph by Zabinsky (2003), 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 getting at each step better offspring and 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, this result holds also for a number of relaxations of PAS that are closer to implementable algorithms, for instance, hesitant adaptive search (Zabinsky, 2003). 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).

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 randomised 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 (1) 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), and (2) 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* $L_0, \ldots, L_q$ *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 $1/\Pr(L_1 \mid L_0) + 1/\Pr(L_2 \mid L_1) + \cdots + 1/\Pr(L_q \mid L_{q-1})$. 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: $\Pr(L_0) \cdot \Pr(L_1 \mid L_0) \cdot \cdots \cdot \Pr(L_q \mid L_{q-1})$, which is upper-bounded by s^q. The expected hitting time of PRS is hence at least s^{-q}. □

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 that 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).

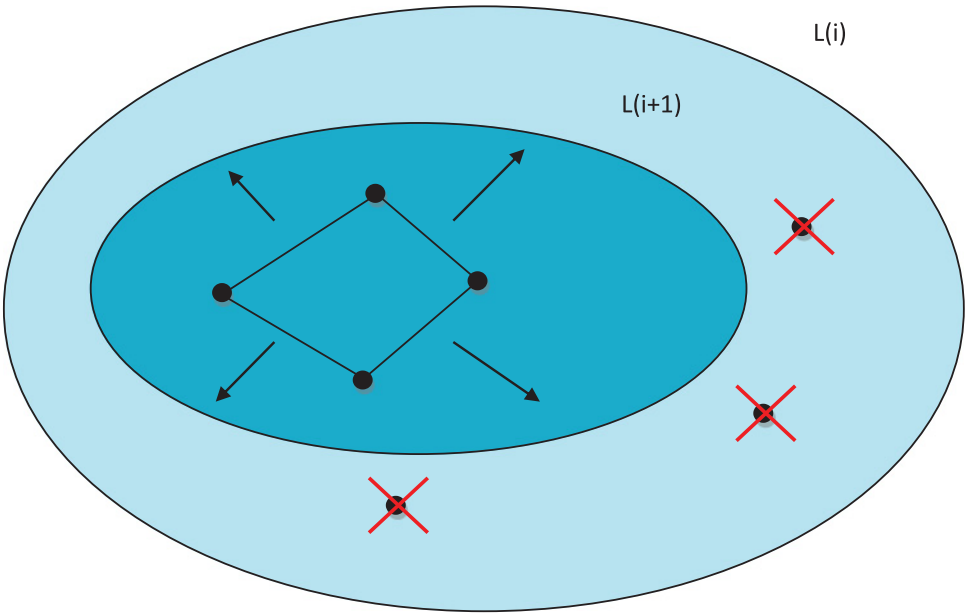

Figure 5: Convex search algorithm dynamics on level sets.

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 large.[6] 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 nonoptimal 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 because 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 $co(P) \supseteq C$.

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 $P_M^{Cov}(m)$ as the worst-case probability that the convex hull of the set P of m*

[6]On 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.

points drawn uniformly at random from a geodesically convex set $C \in \boldsymbol{C_M}$ covers entirely the set C:

$$P_M^{Cov}(m) = \min_{C \in \boldsymbol{C_M}} Pr\left(co(P) = C \mid P = U_m(C)\right).$$

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

$$\forall m' \geq m : \mathrm{P}_M^{\mathrm{Cov}}(m') \geq \mathrm{P}_M^{\mathrm{Cov}}(m).$$

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

LEMMA 4: *Assume fitness levels $L_0, \ldots, L_q$ 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 $co(P'_t) = L_i$. Let the new population P_{t+1} result from μ-times metric convex hull uniform recombination of P'_t, and let $P'_{t+1} = sel(P_{t+1})$ be the next parent population. The probability that $co(P'_{t+1}) = L_j$ for some $i < j \leq q$ is at least*

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

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 $\mathrm{sel}(P_{t+1}) \subseteq L_j$. The probability of covering L_j is at least $\mathrm{P}_M^{\mathrm{Cov}}(\frac{\mu r}{4})$, 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 (Motwani and Raghavan, 1995). The expected number of offspring in L_j is $\mu|L_j|/|L_{j-1}| \geq \mu r$. The probability that fewer than $\mu r/4$ offspring fall in L_j is at most

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

The claim then follows by the union bound. □

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

THEOREM 5: *Assume a quasi-concave fitness function on any metric space M with fitness levels $L_0, \ldots, L_q$, 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*

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

5.2 Boolean Spaces and 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; see Section 3.2).

We first bound the probability of covering any set.

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

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

PROOF: For every geodesically convex set $C \in \mathbf{C_M}$ for the specific case of $M = (S, \mathrm{HD})$ 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 $(1 - 2^{-m+1})^n$. □

Then the bound from Theorem 5 simplifies as follows.

THEOREM 7: *Let $M = (\{0,1\}^n, HD)$ 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*

$$\left(1 - 2^{-\frac{\mu r}{4}+1}\right)^{n(q+1)} - q \cdot exp\left(-\frac{9\mu r}{32}\right)$$

$$\geq \; 1 - (2q+3)n \cdot 2^{-\frac{\mu r}{4}},$$

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. □

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 (Sudholt, 2013). Lehre and Witt (2012) show that the same holds for all unary unbiased black box algorithms, whereas binary black box algorithms run in time $O(n \log\log n)$ (Afshani et al., 2013).

5.3 Integer Vectors and 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 $S_d = \{0, 1, \ldots, d-1\}^n$. 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, for instance, 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 a 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 dimensionwise sum of (unidimensional) 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 unidimensional 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 noncoinciding integers (i.e, nodes) are at distance 1 (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 hyperboxes).

Segment. For (S_d, HD), unidimensional segments are edges, that is, 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. For example, for (S_d, HD), the segment between $(0, 2, 1)$ and $(2, 2, 0)$ is 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 vectorwise aggregation of geometric crossovers acting on each dimension, each one defined on the distance associated with each dimension (Moraglio and Poli, 2006). 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 componentwise 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 subcliques 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 $\{0, 1, \ldots, d-1\}^n$. For example, 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), as for the Hamming space on binary strings, convex sets coincide with segments, that is, they are integer hyperboxes. 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, for example, fixing $*_{012}$ to $*_{01}$. For example, the function LeadingTwos defined on strings with alphabet $\{0, 1, 2\}$ that counts the number of leading 2s in the string is quasi-concave on (S_3, HD), as it can be built by piling up the generalised schemata $*_{012}\,*_{012}\,\ldots *_{012}, 2\,*_{012}\,\ldots *_{012}, 22\ldots *_{012}, \ldots, 22\ldots 2$. For (S_d, MD), piling up a nested chain of convex sets translates into piling up hyperboxes from the largest to the smallest. For instance, 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 hyperbox 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 hyperbox 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 $M = (S_d, HD)$ and $m \in \mathbb{N}$ we have*

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

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. □

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 $M = (\{0,1,\ldots,d-1\}^n, HD)$ 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*

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

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 $M = (S_d, MD)$ and $m \in \mathbb{N}$ we have*

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

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 $2n \cdot (1-1/d)^m$. □

THEOREM 14: *Let $M = (\{0,1,\ldots,d-1\}^n, MD)$ 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*

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

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 $M_B = (\{0,1\}^n, \mathrm{HD})$, that is, 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 optimum.[7]

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 $M_B = (\{0,1\}^n, HD)$ 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 $e^{-n^{\varepsilon}}$ and the expected number of restarts needed to find the optimum is at least $e^{n^{\varepsilon}}$.

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

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

The following statements pertain 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 is that population sizes less than $\log n$ (by a constant factor) always lead to inefficient running times.

[7] This 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 straightforward 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.

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 preceding population size threshold would be $\Theta((\log n)/r) = \Theta(n^c \log n)$, hence asymptotically much 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. The following section 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 $r = o(1)$.

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 from 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 $L_1, \ldots, L_q$ 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 $\varepsilon > 0$, this 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 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 $L_1, \ldots, L_q$ 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 $s = \Theta(r)$, $r \geq 1/poly(n)$ and $q = n^{\Omega(1)}$, then there are constants c_l, c_u such that the expected number of generations is bounded as follows.*

(1) If $\mu \leq c_l \log(qn)/r$, the expected running time is at least $e^{n^{\Omega(1)}}$, i.e., exponential in n.

(2) 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 $c_u \log(qn) \geq 4\log(2(q+2)n)$. □

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 not cover 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 $L_1, \ldots, L_q$. 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 $co(P) \subsetneq L_i$ and $co(P) \setminus L_{i+1} \neq \emptyset$, or if for the initial population P_0 we have $co(P_0) \subsetneq L_i$.*

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 $L_1, \ldots, L_q$ 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 $co(P) \cap L_q = \emptyset$.*

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. □

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 $co(P) = L_i$. 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 $co(P') \subsetneq L_{i+1}$ and $co(P') \setminus L_{i+2} \neq \emptyset$ is at least*

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

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

(1) In the next generation at least one offspring is in $L_i \setminus L_{i+1}$ and at least one offspring is in $L_{i+1} \setminus L_{i+2}$.

(2) In the next generation at most $m_i := 2\mu\ell_{i+1}/\ell_i$ offspring are in L_{i+1}.

(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 preceding 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. □

We also need to bound the probability of the convex search algorithm advancing by many fitness levels in a single step. We say that a population P is *on level* i if $\mathrm{co}(P) \subseteq L_i$ and $\mathrm{co}(P) \setminus L_{i+1} \neq \emptyset$. Now assume the current population is on level i and consider the population reached after the 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 the convex search algorithm with population size μ on any metric space with fitness levels $L_1, \ldots, L_q$. Assume that the current population P is on level i. Consider the first population P' where $co(sel(P)) \subseteq L_{i+1}$ holds. Then for every $j > i$ we have*

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

PROOF: A necessary event for $\Pr(\mathrm{co}(\mathrm{sel}(P')) \subseteq L_j)$ 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 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. □

Taking the two previous lemmas together, we get a pessimistic estimation of the number of misaligned populations the convex search algorithm goes through in a run, unless it has prematurely converged to a nonoptimal point.

LEMMA 23: *Consider the convex search algorithm with population size μ on the metric space M_B with fitness levels $L_1, \ldots, L_q$, 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 the convex search algorithm (CSA) converges to a nonoptimal point.*

PROOF: We may assume that for the initial population P_0 we have $\text{co}(P_0) = L_1$ because otherwise, if $\text{co}(P_0) \subsetneq L_i$ 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 $\text{co}(P) \supsetneq L_{ib+1}$ 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 $\text{co}(P) \subsetneq L_j$ 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 toward 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. □

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*

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

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. □

Now we can finally put all these lemmas together to prove the sought 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

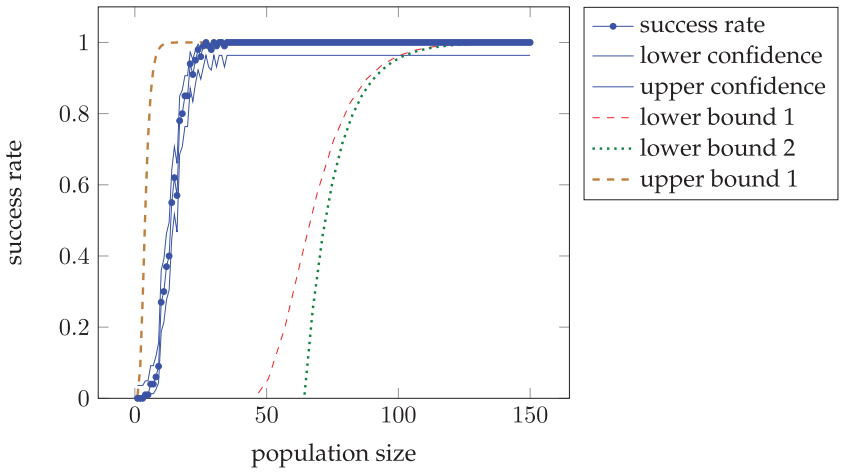

Figure 6: Average success rates for CSA on LeadingOnes with $n = 10$ in 100 runs. The plot also shows the two lower bounds on the success probability from Theorem 7 and the upper bound on the success probability from Theorem 16.

with parameters $\Omega(q/\log(\mu))$ and $\Omega(4^{-\mu s})$. Invoking Lemma 24, the probability of this premature convergence is at least

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

by Bernoulli's inequality. This proves the claim. □

6.3 Experiments for LeadingOnes

We conclude this section with a brief empirical study highlighting the average performance of CSA and comparing it against our theoretical results.[8] 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.

Figures 6 and 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 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. Second, 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

[8]The Python implementation of the algorithm used here is available at https://github.com/amoraglio.

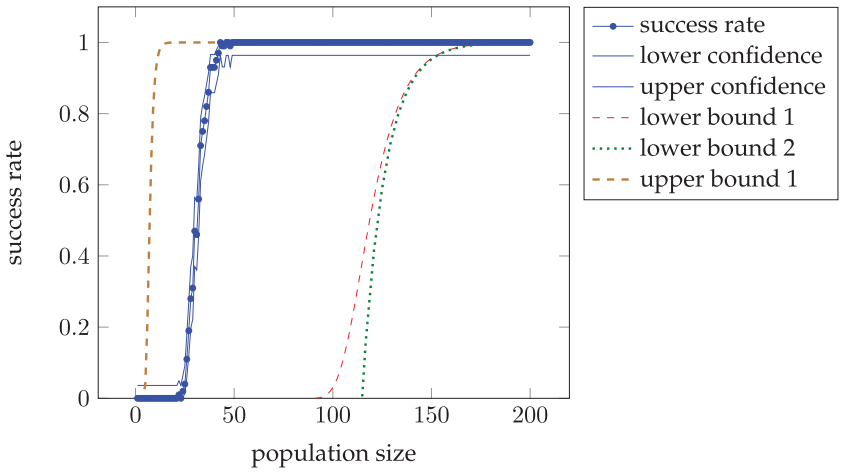

Figure 7: Average success rates for CSA on LeadingOnes with $n = 100$ in 100 runs. The plot also shows the two lower bounds on the success probability from Theorem 7 and the upper bound on the success probability from Theorem 16.

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 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. Similarly, 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 by Böttcher et al. (2010) and by Sudholt (2010). The $(1+1)$ EA is provably the fastest evolutionary algorithm that only uses standard bit mutation for variation (Sudholt, 2013) (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 the expected number of function evaluations of the $(1+1)$ EA on LeadingOnes, which is $\Omega(n^2)$, as shown by Böttcher et al. (2010) and Sudholt (2010). 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 (Sudholt, 2013).

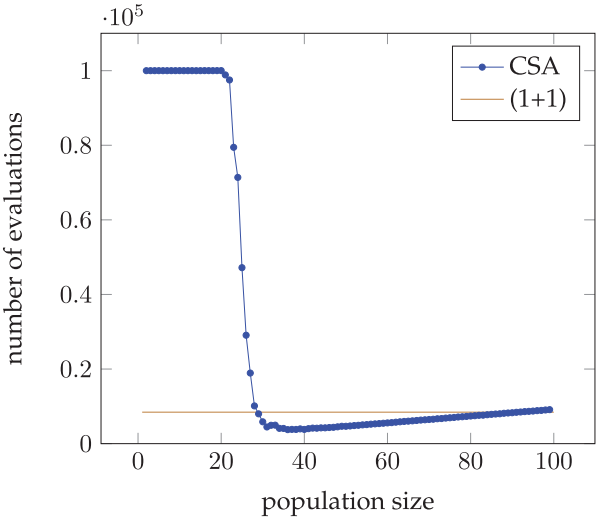

Figure 8: Average number of function evaluations for CSA of 100 runs with varying population sizes, for LeadingOnes with $n = 100$. The number of evaluations was capped at 10^5. The plot also shows the expected number of evaluations for the $(1+1)$ EA as a baseline.

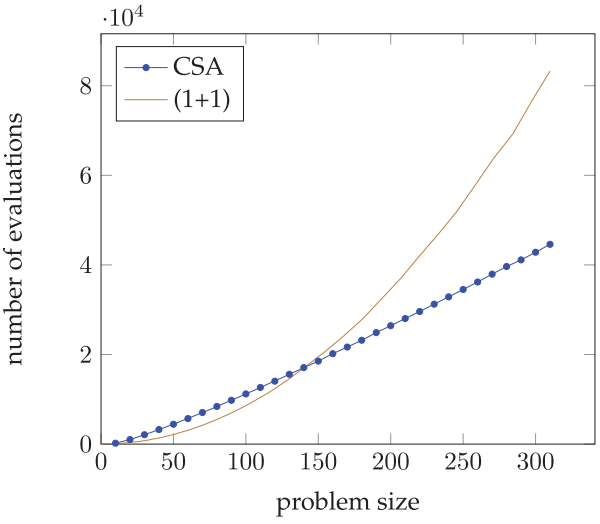

Figure 9: Average number of function evaluations for CSA of 100 runs with increasing problem sizes on LeadingOnes. The plot also shows the expected number of evaluations for the $(1+1)$ EA.

7 Summary and Future Work

Two important open challenges in the evolutionary computation field are (1) finding out on what class of landscapes a certain search algorithm performs well, and why, and (2) devising a more systematic 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 toward 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 it 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 into 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 (Lehre and Witt, 2012; Sudholt, 2013). 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 nonoptimal 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, toward $\mu \leq c_l \log(qn)/r$ 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 that 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 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, parametrised complexity). Finally, as a 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 618091 (SAGE).

References

Afshani, P., Agrawal, M., Doerr, B., Doerr, C., Larsen, K. G., and Mehlhorn, K. (2013). The query complexity of finding a hidden permutation. In A. Brodnik, A. López-Ortiz, V. Raman, and A. Viola (Eds.), *Space-efficient data structures, streams, and algorithms*, pp. 1–11. Lecture Notes in Computer Science, Vol. 8066. Berlin: Springer.

Auger, A., and Doerr, B. (Eds.) (2011). *Theory of randomized search heuristics: Foundations and recent developments*. Singapore: World Scientific.

Böttcher, S., Doerr, B., and Neumann, F. (2010). Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Parallel Problem Solving from Nature*, pp. 1–10. Lecture Notes in Computer Science, Vol. 6238. Berlin: Springer.

Corus, D., Dang, D., Eremeev, A. V., and Lehre, P. K. (2014). Level-based analysis of genetic algorithms and other search processes. In *Parallel Problem Solving from Nature*, pp. 912–921. Berlin: Springer.

De Jong, K. (2006). *Evolutionary computation: A unified approach*. Cambridge, MA: MIT Press.

Doerr, B., Doerr, C., and Ebel, F. (2015). From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87–104.

Doerr, B., Happ, E., and Klein, C. (2012). Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33.

Doerr, B., Johannsen, D., and Winzen, C. (2010). Multiplicative drift analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'10), pp. 1449–1456. ACM.

He, J., and Yao, X. (2004). A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1): 21–35.

Jansen, T. (2013). *Analyzing evolutionary algorithms: The computer science perspective.* Berlin: Springer.

Jansen, T., and Wegener, I. (2002). On the analysis of evolutionary algorithms: A proof that crossover really can help. *Algorithmica*, 34(1): 47–66.

Jansen, T., and Wegener, I. (2005). Real royal road functions: Where crossover provably is essential. *Discrete Applied Mathematics*, 149:111–125.

Kötzing, T., Sudholt, D., and Theile, M. (2011). How crossover helps in pseudo-Boolean optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'11), pp. 989–996. ACM.

Langdon, W., and Poli, R. (2002). *Foundations of genetic programming*. Berlin: Springer-Verlag.

Lehre, P. K., and Witt, C. (2014). Concentrated hitting times of randomized search heuristics with variable drift. In *Proceedings of the International Symposium on Algorithms and Computation*, pp. 686–697. Lecture Notes in Computer Science, Vol. 8889. Berlin: Springer.

Lehre, P. K., and Witt, C. (2012). Black-box search by unbiased variation. *Algorithmica*, 64(4): 623–642.

Mitavskiy, B. S. (2004). A mathematical model of evolutionary computation and some consequences. Unpublished doctoral dissertation, University of Michigan, Ann Arbor.

Moraglio, A. (2007). Towards a geometric unification of evolutionary algorithms. Unpublished doctoral dissertation, University of Essex, UK.

Moraglio, A. (2011). Abstract convex evolutionary search. In *Proceedings of the Workshop on Foundations of Genetic Algorithms*, pp. 151–162.

Moraglio, A., and Poli, R. (2004). Topological interpretation of crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'04), pp. 1377–1388. ACM.

Moraglio, A., and Poli, R. (2006). Product geometric crossover. In *Parallel Problem Solving from Nature*, pp. 1018–1027. Berlin: Springer.

Moraglio, A., and Sudholt, D. (2012). Runtime analysis of convex evolutionary search. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'12), pp. 649–656. ACM.

Motwani, R., and Raghavan, P. (1995). *Randomized algorithms*. Cambridge: Cambridge University Press.

Neumann, F., and Witt, C. (2010). *Bioinspired computation in combinatorial optimization: Algorithms and their computational complexity*. Berlin: Springer.

Patel, N. R., Smith, R. L., and Zabinsky, Z. B. (1988). Pure adaptive search in Monte Carlo optimization. *Mathematical Programming*, 4:317–328.

Radcliffe, N. (1991). Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–205.

Reidys, C. M., and Stadler, P. F. (2002). Combinatorial landscapes. *SIAM Review*, 44:3–54.

Rothlauf, F. (2002). *Representations for genetic and evolutionary algorithms*. Berlin: Springer.

Rowe, J. E., Vose, M. D., and Wright, A. H. (2002). Group properties of crossover and mutation. *Evolutionary Computation*, 10(2): 151–184.

Stephens, C. R., and Zamora, A. (2003). EC theory: A unified viewpoint. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'03), pp. 1394–1405. ACM.

Sudholt, D. (2005). Crossover is provably essential for the Ising model on trees. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'10), pp. 1161–1167. ACM.

Sudholt, D. (2010). General lower bounds for the running time of evolutionary algorithms. In *Parallel Problem Solving from Nature*, pp. 124–133. Lecture Notes in Computer Science, Vol. 6238. Berlin: Springer.

Sudholt, D. (2012). Crossover speeds up building-block assembly. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO'12), pp. 689–696. ACM.

Sudholt, D. (2013). A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 17(3): 418–435.

Surry, P. D., and Radcliffe, N. J. (1996). Formal algorithms + formal representations = search strategies. In *Parallel Problem Solving from Nature*, pp. 366–375. Berlin: Springer.

van de Vel, M. (1993). *Theory of convex structures*. Amsterdam: Elsevier.

Wegener, I. (2002). Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In R. Sarker, X. Yao, and M. Mohammadian (Eds.), *Evolutionary optimization*, pp. 349–369. Berlin: Springer.

Wolpert, D. H., and Macready, W. G. (1996). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1): 67–82.

Zabinsky, Z. B. (2003). *Stochastic adaptive search for global optimization*. Berlin: Springer.