

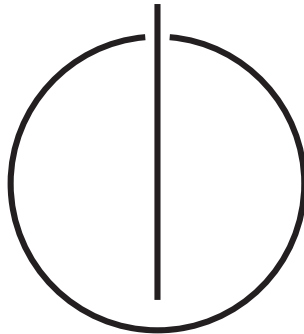
FAKULTÄT FÜR INFORMATIK

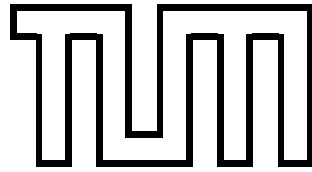
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Generating Swedish-style Crossword Puzzle Masks using Evolutionary Algorithms

Jakob Julian Engel





FAKULTÄT FÜR INFORMATIK

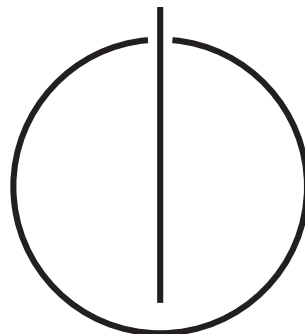
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Generating Swedish-style Crossword Puzzle Masks
using Evolutionary Algorithms

Erzeugen von Schwedenrätselmasken mit Evolutionären
Algorithmen

Author: Jakob Julian Engel
Supervisor: Prof. Dr. rer. nat. habil. Jürgen Schmidhuber
Advisors: Dipl.-Inf. Oliver Ruepp
Dipl.-Inf. Frank Sehnke
Date: September 15, 2009



Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. September 2009

Jakob Julian Engel

Acknowledgments

First of all I would like to thank those who supported me in the process of writing this Bachelor Thesis. Major thanks go to my two supervisors Frank Sehnke and Oliver Ruepp. This thesis is largely based on Oliver's study project on crosswords [?], and without his advice about the design of crossword masks the results would probably not have been nearly as good. Frank I thank for introducing me to evolutionary algorithms and for his very helpful suggestions while writing this thesis.

Furthermore I would like to thank Markus Holzer for coming up with the idea to use evolutionary algorithms to generate crossword masks in the very first place.

Contents

Abstract	xi
Zusammenfassung	xiii
1. About Genetic Algorithms	1
1.1. General Functionality	1
1.2. Selection	2
1.3. Mutation	4
1.4. Crossover	5
1.5. Diversity vs. Convergence	6
1.6. Summary	7
2. About Crossword Masks	9
2.1. Basic Definitions	10
2.2. Validity Constraints	12
2.3. Quality Criteria	12
3. Applying a Genetic Algorithm	15
3.1. General Setup	15
3.2. The Fitness Function	15
3.2.1. Coverage	16
3.2.2. Word Lengths	16
3.2.3. Clustering	17
3.2.4. Invalid Definition Fields of Type 3,4,5,6	17
3.2.5. Dead Ends	18
3.2.6. Result	18
3.2.7. Localized Fitness	18
3.3. Initialization	19
3.4. Mutation	20
3.5. Crossover	24
3.6. Results	27
3.7. Discussion	31
4. Memetic Algorithm Approach	35
4.1. Basic Idea	35
4.2. Implementation	36
4.3. Results	38
5. Practical Results	41

6. Further Work	43
Appendix	47
A. Code	47
Mutation	47
Crossover	48
Basic Hillclimber	49
Basic Genetic Algorithm	50
Memetic Approach	52
B. Sample Masks	53

Abstract

This Bachelor Thesis addresses the problem of generating a so-called *mask* for Swedish-style crossword puzzles using evolutionary algorithms. Such a mask defines only the general layout of a crossword, i.e. the placement of words and word-descriptions within the rectangular grid: basically a mask assigns a specific field type to each grid cell. This however is subject to several complex constraints and quality criteria, which first have to be (and in this thesis are) formulated. In current practice when creating a professional crossword puzzle, such a mask is usually still created *by hand* - as no algorithm capable of creating adequate masks exists.

The problem can be seen as a constrained optimization problem over a large (exponential in the number of grid cells, 7^{400} for a mere 20×20 mask), discrete search space, where neither constraints nor the characteristic to be optimized can be formulated in a compact form. As a globally optimal solution is neither required, nor can be found efficiently, *evolutionary algorithms* as a heuristic optimization technique without need for a mathematical description of the problem are a sensible choice.

First, a *genetic algorithm* for solving the problem is developed and implemented. It will however be shown - by both, practical tests and theoretical considerations - that due to extremely *strong local dependencies* between nearby field assignments, the crossover operator is more destructive than beneficial and ultimately the complete genetic algorithm is outperformed slightly by a simple *hillclimber*.

In order to still be able to exploit the fact that globally, different areas of a mask hardly affect each other at all, a *memetic algorithm* is developed using crossover as single operator and applying a hillclimber afterwards to each resulting individual in order to repair any damage resulting from the crossover, hence fully exploiting the solution. This approach performs significantly better than both, a simple hillclimber and a complete genetic algorithm - and generates masks that can compete with manually created ones.

Zusammenfassung

Die Bachelorarbeit befasst sich mit dem automatischen Generieren von Kreuzworträselmasken für Schwedenrätsel mittels evolutionärer Algorithmen. Eine solche Maske legt lediglich das generelle Layout eines Rätsels fest, d.h. die Position und Art von Angabefeldern - nicht jedoch die im fertigen Rätsel einzutragenden Wörter. Diese Zuweisung unterliegt einer Reihe von Gültigkeits- und Qualitätskriterien, welche zuerst extrahiert und formuliert werden müssen. In heutiger Praxis wird eine solche Maske üblicherweise per Hand erstellt, da automatisch generierte Masken qualitativ minderwertig sind.

Diese Aufgabe kann als beschränktes Optimierungsproblem über einem großen (exponentiell in der Anzahl der Gridzellen) diskretem Suchraum gesehen werden, in dem weder die Nebenbedingungen noch die zu optimierende Größe explizit formuliert werden kann. Da eine global optimale Lösung weder benötigt wird noch effizient gefunden werden kann, stellen evolutionäre Algorithmen einen idealen Lösungsansatz dar.

Anfangs wird ein Genetischer Algorithmus entwickelt und implementiert. Es stellt sich jedoch heraus, dass, aufgrund starker Abhängigkeiten zwischen nahe liegenden Feldzuweisungen, der Crossover Operator keine Vorteile bietet, und letzten Endes liefert ein simpler Hillclimber bessere Ergebnisse als ein kompletter Genetischer Algorithmus.

Um dennoch die Vorteile von Crossover ausnutzen zu können wird ein Memetischer Algorithmus entworfen, welcher ausschließlich auf Crossover und nachfolgender "Reparatur" der entstehenden Masken mittels eines Hillclimbers basiert. Dieser Ansatz stellt sich als deutlich besser als ein Hillclimber oder Genetischer Algorithmus alleine heraus. Die durch diesen Ansatz generierten Masken können qualitativ mit Hand gemachten konkurrieren.

1. About Genetic Algorithms

A genetic algorithm is a heuristic optimization technique used in computer science. As part of the more general class of evolutionary computation, it is motivated by the process of natural evolution, exploiting the principle of “survival of the fittest”.

The most important characteristic of evolutionary computation in general is, that very little problem-specific domain knowledge is required: One only needs to somehow measure how good a given solution is (either absolute, or relative to other given solutions by, for example, letting them compete against each other). Additionally, a suitable representation for solutions in the solution domain has to be found.

One however does not require any further information concerning - for example - some gradient, as most other techniques do. On the other hand, using techniques which exploit such information often gives better practical results for problems where such information is available - simply due to the fact that in these cases evolutionary algorithms do not exploit all the problem knowledge available.

Genetic algorithms in particular apply these principals to discrete problems, where solutions traditionally are represented as a binary string of 0s and 1s of fixed length. Arguably, this does not impose any real limitations - as in computer science basically everything is represented as such a binary string.

In order to achieve good results though, the binary string should represent a fixed number of (discrete) features. Furthermore, a high independence between these features, or at least between clusters of features, is desirable.

In this chapter the basic methods of genetic algorithms: selection, mutation and crossover will be explained, and an overview over the general functionality will be given.

1.1. General Functionality

Rather than attempting to simulate the whole process of natural evolution, genetic algorithms extract only the most basic principles of this process and use them to solve a given optimization task.

The basic algorithm operates on a fixed-size **population** of **individuals**, each representing one solution. The representation of an individual, or rather of one feature of an individual is called **genotype**, while its behavior interpreted with respect to the given problem is called **phenotype**.

In a first step, each individual in the current population (current **generation**) is evaluated by the means of a **fitness function**, and a subset, favoring the better individuals, is selected. In a second step, the selected individuals are used as basis for creating the next generation by the means of **mutation** and **crossover**. Starting with a randomly generated first generation, this process continues until some break condition is satisfied.

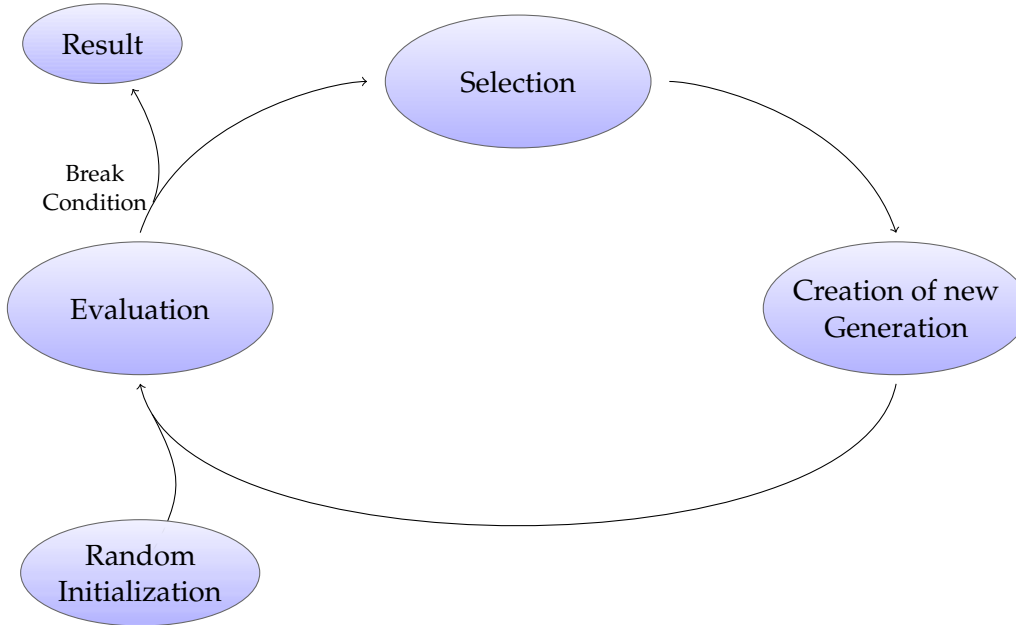


Figure 1.1.: Schematic representation of the Algorithm

Of course this is only a very general description of the algorithm. It can and in practice is often heavily modified. For example rather than selection and creation of the next generation being separate steps, in practice for each individual to be created, one (in case of mutation) or two (in case of crossover) “parents” are selected by the means of some (probabilistic) **selection function**. This Function can be modeled as distribution over a discrete random variable X denoting the selected individual, with the parent generation as domain.

1.2. Selection

As the selection is a vital part of evolutionary algorithms, the choice of the selection strategy and the respective parameters can highly influence the results.

The most important aspect of the selection strategy is the **selection pressure**, determining how much the fitness value of an individual influences it’s chance to be selected. While a high selection pressure heavily favors better individuals and hence leads to increased convergence speed, a low selection pressure basically gives solutions with a lower fitness value a better chance to reproduce as well, increasing the breadth of the search and

hence giving a better chance to avoid local optima.

Furthermore, one can choose to strictly separate the generations, i.e. only to allow newly created individuals in the next generation (called (μ, λ) - strategy, creating λ new individuals by applying mutation and crossover to the best μ individuals of the parent generation). Alternatively the new generation can be composed of both, individuals from the parent generation which are just copied to the new generation, and newly created individuals (called $(\mu + \lambda)$ - strategy, creating λ new individuals and copying μ). A popular method is to copy only the best individual to the next generation without changing it, while the rest of the new generation consists only of newly created individuals.

In practice, one very popular selection function is the so-called **tournament selection**: One first randomly determines α individuals (using a uniform distribution), compares them, and selects the one with the highest fitness value.

The two simple advantages of this method are, that the selection pressure is easily controllable by choice of α , and that it is very simple to implement¹. Note that this method basically translates to a probability distribution over the parent generation which only depends on α and the *position* of an individual if the population is sorted by fitness - and not on the actual *fitness values*.

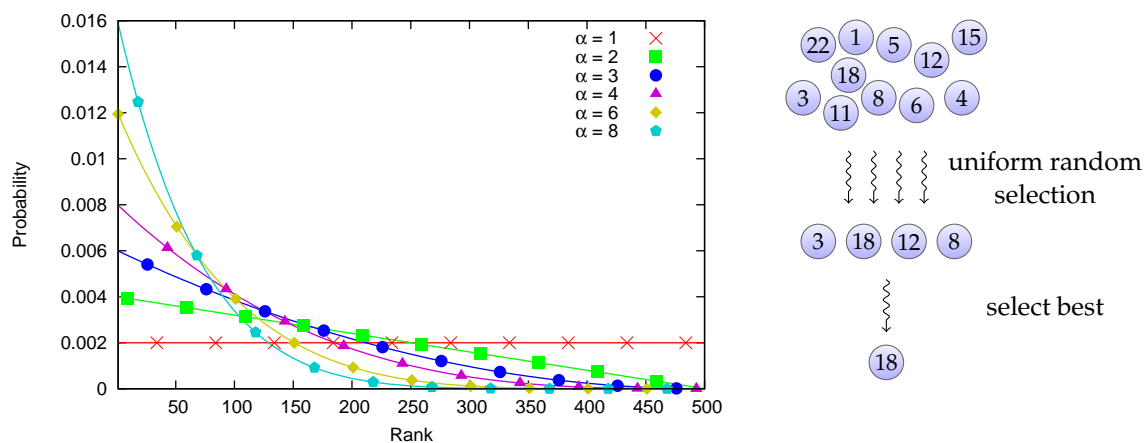


Figure 1.2.: Selection probabilities using tournament selection for a population size of 500. The probability for an individual to be selected is given by $Pr[r] = \frac{(n-r+1)^\alpha - (n-r)^\alpha}{n^\alpha}$, where $n \in \mathbb{N}$ is the population size and $r \in [1; n]$ the position of the individual. Note that an individual can participate multiple times in the same competition and hence even the worst individual has a non-zero probability of being selected (although, obviously, a very small one).

Another possible selection function is the so called **roulette-wheel selection**: Here, for each individual the probability to get selected is directly proportional to its fitness value. This approach has the often desirable effect that big improvements are taken over fairly quickly, while in generations where only small improvements are discovered, the search is broadened automatically. In practice though, the overall selection pressure is

¹Pseudocode for this particular selection function can be found in appendix A

difficult to control and highly dependent on the fitness function itself, which might pose problems².

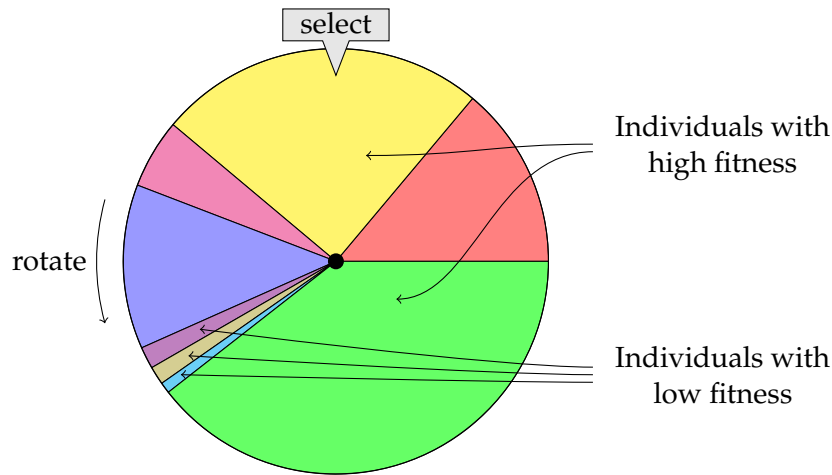


Figure 1.3.: Visualization of roulette-wheel selection. Each individual is assigned a share of the wheel proportional to its fitness value. The wheel is then rotated and the position it stops in determines the individual to be selected.

1.3. Mutation

Mutation poses one of the two main operators used in order to create the new generation.

Traditionally on a bit string-level, mutating an individual simply means flipping one or more bits, the result being a slightly modified version of the parent individual. This can be done by randomly choosing k bits to be changed, or by specifying a mutation-probability λ deciding for each bit whether it is flipped or not, resulting in an average of $\lambda \cdot n$, n denoting the length of the bit string, bits changed. The number of bits changed is called **mutation step size**.

On a higher abstraction level there are obvious other possibilities - for example adding a (rounded) $\mathcal{N}(0, \sigma^2)$ -distributed random number to some integer value, or making some other problem- and representation-specific adjustments.

In general, mutating an individual should produce a (with respect to the given problem, i.e. the phenotype of the individual) similar, but slightly different - and hopefully better - child-individual. Equally, the more alike two individuals (again, with respect to their phenotypes) are, the greater the probability for one to be the result of mutating the other should be. This is a very important point as often some adjustments are needed to achieve it: even in a very basic setup using individuals consisting of only one binary encoded

²This problems can partly be diminished by appropriate normalization; whereas the selection pressure can be controlled by raising the normalized fitness values to some power $p \in \mathbb{R}^+$ ($p < 1 \rightarrow$ less pressure; $p > 1 \rightarrow$ more pressure) - still this often does not suffice

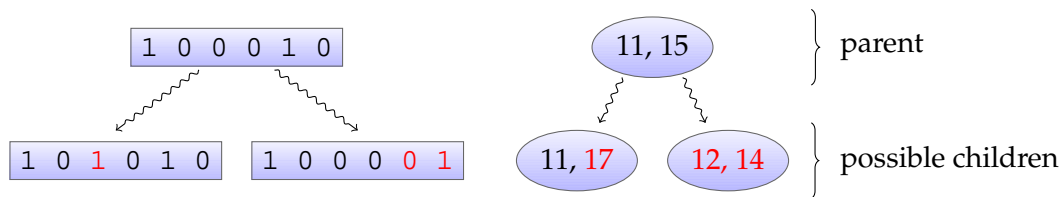


Figure 1.4.: Schematic representation of mutation.

integer, the genotypes (“representation”) of two individuals might be very similar while their phenotypes (“behavior”) are fairly different ($000000 = 0$ and $100000 = 32$). On the other hand, two individuals having very different genotypes might be quite similar with respect to their phenotype ($100000 = 32$ and $011111 = 31$)³. This issue and the consequence that, in this case a simple bit flip-mutation is very unlikely to find the, in terms of phenotypes, very small step from 31 to 32 is widely known as **Hamming Wall**. It is to mention that the later situation, i.e. very different genotypes for similar phenotypes, can have far worse consequences than the other way round.

1.4. Crossover

The crossover operator basically takes two parent individuals, combines them and produces a child individual, simulating natural sexual reproduction. On a bit string-level, this can be done by determining one or more splitting points and using approximately half of each parent individual, as depicted in figure 1.5.

On a higher abstraction level again, there are other possibilities. Different numerical values can, for example, be interpolated (i.e. averaged), or even extrapolated (i.e. if 37 gives a good result, and 40 an even better result, one might try 42 for obvious reasons) - although admittedly the later has little to do with natural sexual reproduction.

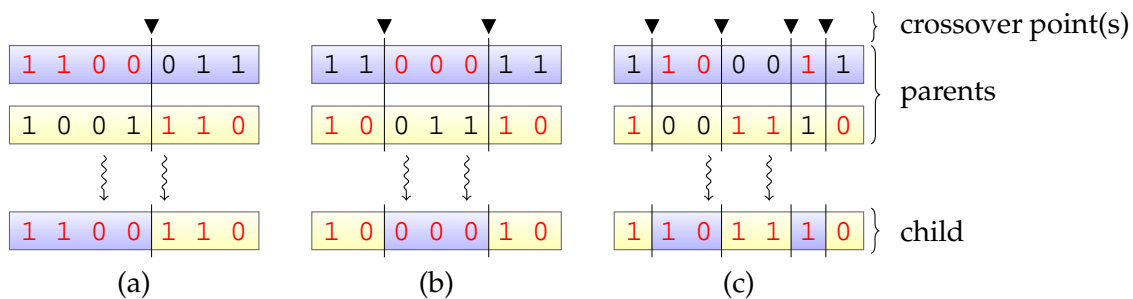


Figure 1.5.: Schematic representation of crossover on a bit string-level. (a) one-point-crossover (b) two-point-crossover (c) n-point-crossover

The value of this operator mainly lies in the assumption that different parent individuals might have different strengths and weaknesses, and that with a little luck the better

³for this simple example, the problem could be greatly reduced by using a Gray-code representation, or by performing mutation on a higher abstraction level in the first place.

part of each parent combined results - at least in some cases - in an even better child. It also enables individuals to adopt improvements found by some other individual, while retaining at least a part of their original form. Arguably this base assumption can - for some problems - be problematic: If strong dependencies between different parts of an individual exist, ripping it apart at a random point is likely to impair both halves, hence making the produced child useless. Similarly the two halves combined might just not be "compatible", leading to the same result.

As most dependencies cannot be avoided⁴, one simple way to reduce this problem is to design the representation of an individual in such a way that highly dependent features are closely together, while more independent features are at a bigger distance. Using one-point- or two-point-crossover then minimizes the probability for these complications to occur.

Still, even concerning problems where these problematics arise, the crossover operator has proven to be very valuable, and using it can, for most problems, improve the result significantly. There also are several methods designed to deal with these issues, for example by trying to "remember" good crossover points, i.e. points at which a crossover is least likely to have negative effects. [?].

There in fact exists some theoretical work on this subject⁵, but, primarily due to the high dependence on the actual problem, it is hardly of practical relevance - in fact even it's theoretical justification is arguable [?].

1.5. Diversity vs. Convergence

One of the major strengths of evolutionary algorithms lies in the ability to explore the search space very exhaustively. While for example most gradient-based approaches tend to immediately run towards the closest extreme point - which for complex problems is likely to be a mere local optimum - evolutionary methods in general have far better chances to overcome such local extreme points. This is due to the simple fact that many areas of the search space are explored simultaneously by the different individuals, while the crossover operator allows improvements found by one individual to be - if compatible - adopted by others; hence making the whole process more efficient than a number of independent, local searches.

In order to fully exploit this strength however, a certain diversity within each generation has to be assured, i.e. the individuals should not be too similar, such that they actually explore different areas of the search space (*exploration*). On the other hand, when keeping the individuals well distributed, existing good solutions might not be exploited fully, leading to extremely slow (or even stagnant) convergence (*exploitation*).

This so-called **Exploration versus Exploitation Dilemma**, is one of the most significant problems to be dealt with; especially premature convergence towards a local optimum is often difficult to avoid. The most obvious and direct way to influence this is the selection

⁴otherwise the whole problem could be split into two separate optimization problems

⁵see *Building block hypothesis* [?]

pressure, i.e. giving less good individuals a better chance to be selected - this however is only practicable to a certain degree, as it drastically decreases the convergence speed.

There exist a number of modifications to deal with this issue: One possibility for example is to include the average dissimilarity to other individuals as additional criterion for determining the fitness value (i.e. the more different compared to the other individuals an individual is, the better its fitness value). Another possibility is to only allow individuals that have a certain dissimilarity to all other individuals within the population (usually the minimal dissimilarity is decreased over time). Obviously some kind of metric needs to be defined in order to implement these approaches - for bit strings the Hamming distance is an obvious choice.

1.6. Summary

In general it can be said that for some problems evolutionary algorithms give much better results than any other method applicable. Exactly which problems "some" are, and exactly how good "much better" is, often depends on design choices and the implementation itself. Especially for complex problems that are difficult - or impossible - to trace mathematically as the problem presented in this thesis, evolutionary algorithms perform very well.

Due to the fact that the whole process is based on heuristics and not on (practically relevant) mathematical theory⁶, there is no absolute truth concerning design choices⁷. One rather has to find a suitable setup for a specific given problem, keeping in mind that seemingly small or unimportant aspects, such as the way a solution is encoded, might greatly affect the outcome.

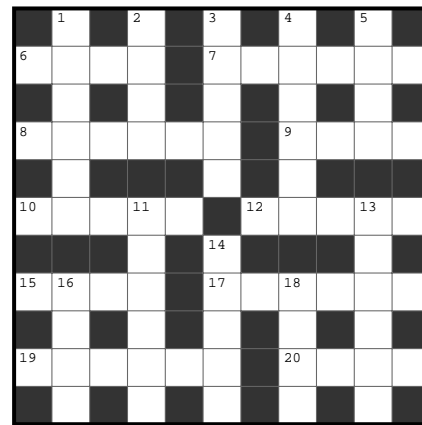
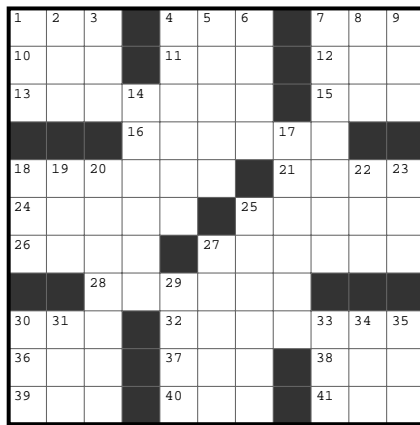
Furthermore there are numerous modifications not discussed at this point, which can help in solving or reducing specific problems. For further information one can refer to corresponding literature, for example [?].

⁶as already mentioned above, such theory does exist - in addition to the above one can refer for example to Holland's schema theorem [?] - but the *practical* value of these is highly questionable.

⁷there are however several heuristics not discussed here, for example the so-called (1/5-th) success rule - still whether such heuristics apply is, again, highly dependable on the actual problem.

2. About Crossword Masks

Crossword puzzles are said to be the most popular and widespread word game in the world, yet they have a short history. While an early predecessor of crossword puzzles appeared in England as early as in the 19th century, the puzzle in its today common form has its origin in the USA, where the first of its kind was published in the *New York World* in 1913 [?].



Haut- erkrankung	Verkehr, Gesell- igkeit	Küchen- gerät	umgangs- sprachl.: leichter Stoß	Glöck- chen, Klingel	Untugend	Geburts- stadt des heiligen Franz	latei- nisch: ich
↳	▼	▼	▼	▼	▼	▼	gedeckter Tisch
persön- liches Fürwort (4. Fall)	▶			unrett- barer Wüstling (griech.)	▶		▼
münd- licher Befehl	▶					Europ. Weltraum- organis. (Abk.)	▼
↳				Zone niedrigen Luft- drucks	▶		
Haupt- stadt von Afgha- nistan		Nackt- heit	▶				
mittel- deutsch: obergäri- ges Bier	▶			Währungs- einheit im Iran	▶		

(c)

Figure 2.1.: (a) American-style grid. (b) English-style grid. (c) Swedish-style grid (German).

Today, a variety of different crossword puzzle styles exist, some of which are shown

in figure 2.1. In this work however we will focus only on the so called **Swedish-style crossword puzzles**. Such a puzzle is presented as a rectangular grid consisting of three different types of fields: **definition fields**, **letter fields** and **cut-out fields**. The task of the puzzle solver is to guess the words that are described by the definition fields and to fill out the corresponding letter fields that are denoted by an arrow from the definition field.

In current practice, crossword puzzles are created in two steps: At first, a so-called *mask* is created, denoting only the arrangement of letter fields and definition fields. The actual words filling the puzzle are then found in a second step, resulting in the complete crossword puzzle. The reason for this separation is that, while efficient computer programs exist for solving the second step, the first step is still often done manually. Although there in fact exist computer programs which can create valid masks, those are usually of inferior quality. [This introduction text, the crossword puzzle displayed in 2.1 (c) and the following definitions of letter field, cut-out field and definition field are partly taken from [?].]

2.1. Basic Definitions

- Mask:** A mask is a two-dimensional rectangular grid, where each field may be either a **letter field**, a **definition field** or a **cut-out field**. By introduction of a set of control characters $S := \{0, 1, 2, 3, 4, 5, 6, \#\}$, where 0 represents a letter field, # represents a cut-out field and 1 to 6 represent the different types of definition fields, a mask can naturally be represented as matrix $M \in S^{n \times m}$.
- Letter field:** A letter field simply denotes a blank space, which the solver of the crossword puzzle is to fill in.
- Definition field:** Definition fields denote the fields where the word-descriptions are printed in. In current crossword puzzles several types of definition fields can be found, including so-called double definition fields and definition fields covering two adjacent grid cells. In this thesis however, only the most common six types of definition fields are allowed, as depicted in figure 2.2.
- Cut-out field:** Cut-out fields denote fields that are not part of the actual crossword puzzle; usually they reserve space for pictures or solutions of former puzzles.
- Word:** Each definition field defines exactly one word. The word starts at a field dependent on the type of its definition field, and continues along the corresponding row or column, until either the end of the grid is reached, or the next field is not a letter field. Words defined by a definition field of type 1,5 or 6 are called horizontal words, words defined by a definition field of type 2,3 or 4 are called vertical words.

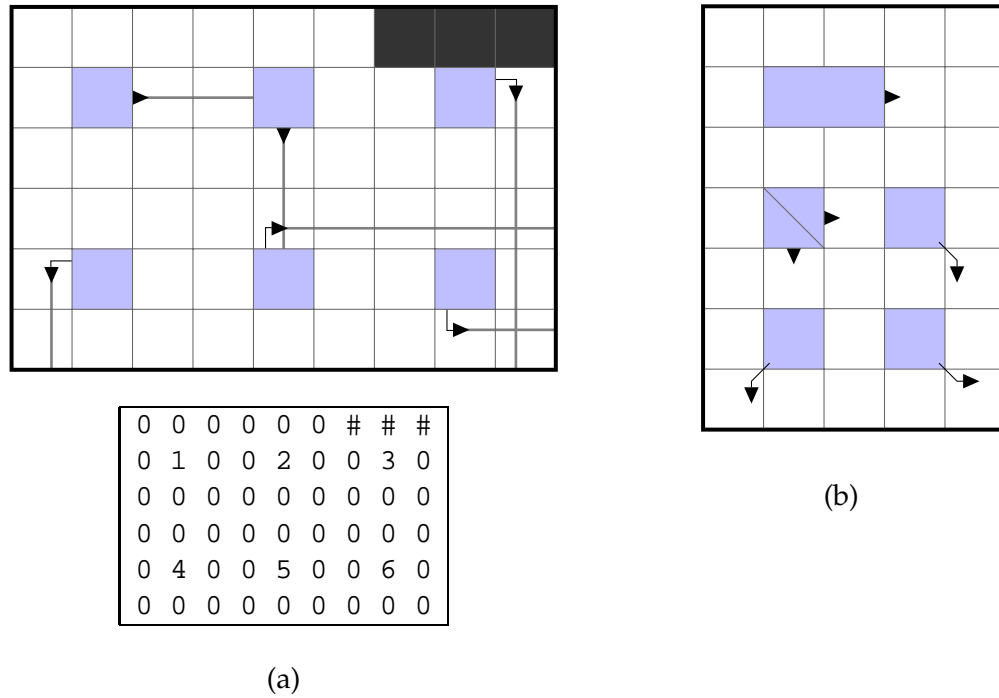


Figure 2.2.: (a) different field types allowed and the corresponding matrix M . The gray lines illustrate the words defined by the corresponding definition fields. (b) other possible definition field types not considered in this work.

2.2. Validity Constraints

It is obvious that not every possible matrix $M \in S^{m \times n}$ corresponds to a valid crossword mask. We therefore define four simple, absolute constraints which a valid mask has to meet:

1. each letter field is to be part of at least one word.
2. each word has to span over at least two letters. A definition field which does not have any corresponding letter fields simply defines a word of length zero.
3. each word is to be enclosed in between two non-letter fields.
4. no two horizontal or two vertical words may overlap.

Examples for the violation of these four constraints are shown in figure 3.2. Note that at this point a letter field is not required to be covered horizontally *and* vertically, in fact there are valid masks where no two words intersect.

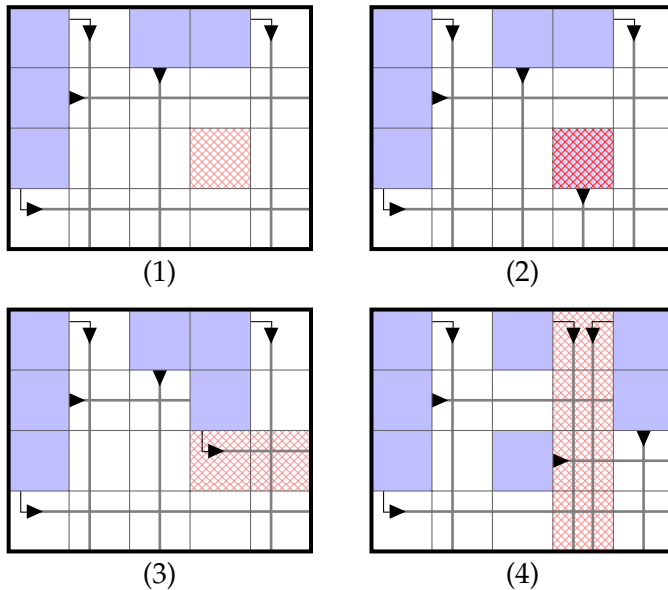


Figure 2.3.: Violations of the four Validity Constraints.

2.3. Quality Criteria

In order to be of practical value, the validity of a mask is by far not enough. As the name strongly suggests, in a crossword puzzle it is desirable that words actually cross, such that finding the solution for one word gives hints towards other words. Furthermore, the layout of the mask should be appealing to the human solver - as this is mainly a subjective criteria, it can not be defined explicitly, however several heuristics can be formulated.

In addition to the above, it needs to be possible to find a number of words to fill the crossword puzzle, such that every letter field is uniquely defined. Some basic criteria for a “good” mask are the following:

1. **Coverage:** Naturally, a large number of horizontally *and* vertically covered fields is desired.
2. **Word lengths:** the optimal word length is four to six letters. Longer words are usually more interesting for the solver, while on the other hand it becomes difficult to find fitting words with more than eight letters; hence words with more than eight letters should not occur frequently.
3. **Clustering of definition fields:** in Swedish-style crossword puzzles, large “clusters” of definition fields are to be avoided, such that definition fields and letter fields are distributed as evenly as possible.

Especially long chains of adjacent definition fields are undesirable, as are so-called “dead ends” where the first or last letter of a word is enclosed by three non-letter-fields (excluding the ones at the left and top border, as there such situations are unavoidable).

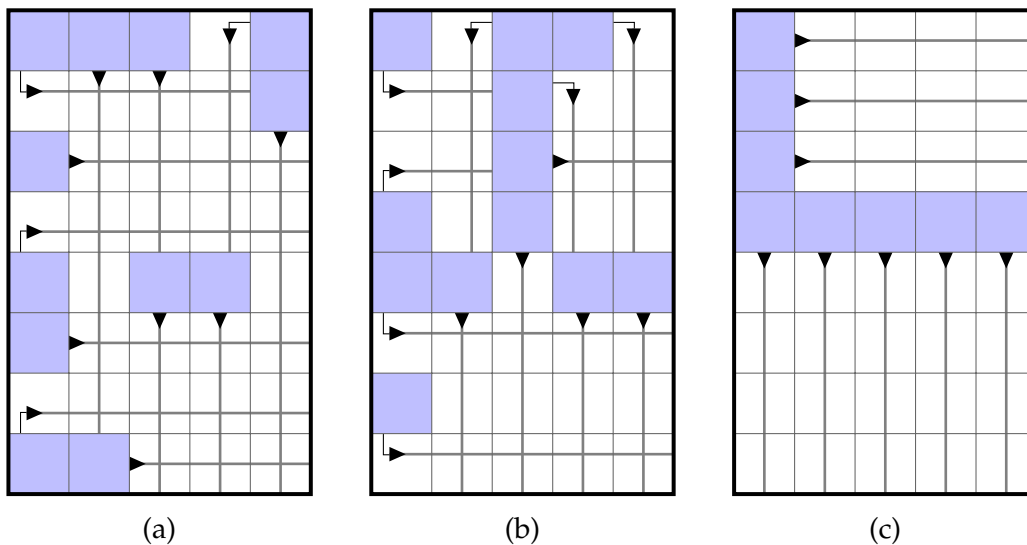


Figure 2.4.: All three masks shown are, according to the definition above, valid masks. Mask (a) performs quite well concerning the quality criteria: No clusters with more than 3 definition fields exist, all word lengths are between three and six and a maximum coverage is achieved. Mask (b) however performs significantly worse: There are three words with only length two, a cluster of size ten splitting the whole grid in three parts, several fields which are only covered once and two “dead ends”. Mask (c) performs even worse: As no two words intersect, it can hardly be called a crossword puzzle.

Arguably the distinction between validity constraints and quality criteria is somewhat arbitrary. For example one could demand that, for a mask to be valid, at least 75% of the

letter fields are to be covered both, horizontally and vertically, or that all letter fields must be 4-connected, i.e. the mask is not split apart by definition fields.

The motivation for the above definition of “valid” is to find a minimal set of easily expressible and simple to compute absolute properties, ensuring that the mask could (at least theoretically) be used for a crossword. It should be clear that generating valid masks according to this definition is trivial (see Figure 2.4 (c)), and hence the challenge lies in creating valid masks which perform as good as possible on the quality criteria.

3. Applying a Genetic Algorithm

3.1. General Setup

The goal of this work is to generate a valid mask performing as good on the quality criteria as possible, such that it can be used for a crossword puzzle. The desired dimensions of the mask, as well as the positions of cut-out fields are given at the beginning and must not be changed - of course the resulting algorithm should be applicable for generating masks with any reasonable dimensions and cut-out field positions.

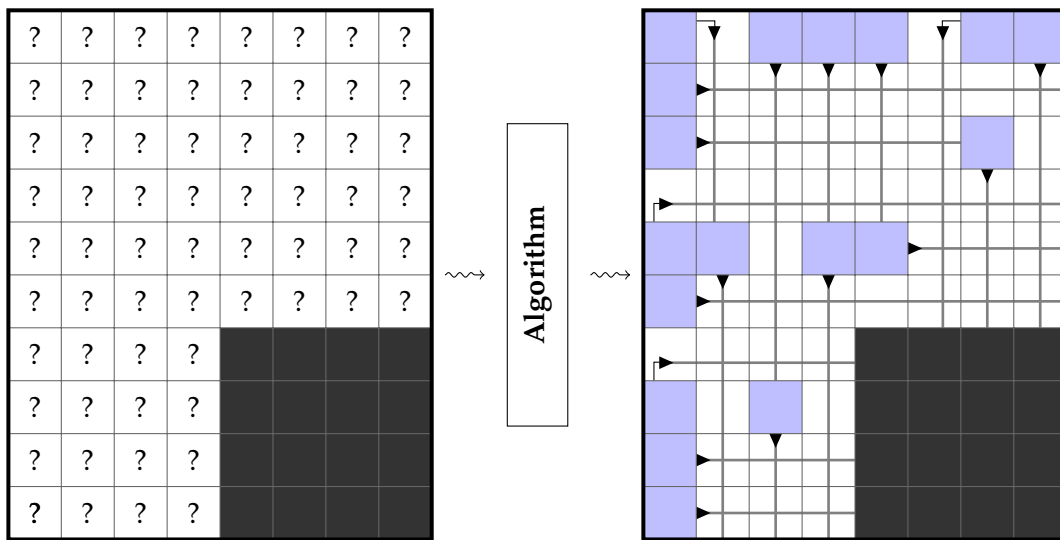


Figure 3.1.: Task of the algorithm.

Each mask corresponds to exactly one individual. A mask is represented simply as Matrix (i.e. two-dimensional array) $M \in S^{n \times m}$ with $S := \{0, 1, 2, 3, 4, 5, 6, \#\}$, as introduced in the previous part.

3.2. The Fitness Function

It has been decided not to treat validity and quality of a mask as separate goals, but rather to optimize both aspects simultaneously. This is justified by the fact that they are highly interdependent: optimizing only one criterion very quickly destroys achievements concerning the other one. On the other hand, a big progress concerning one criterion might be worth small sacrifices with respect to the other. Furthermore, it is very difficult to design operators which operate solely on valid masks - every kind of repairing algorithm either

heavily relies on trial & error, or has to cover hundreds of situations and hence be very complex.

This is done by accumulating *penalty points* for several criteria, some of which correspond directly to the validity constraints, others are simple and easy to compute heuristics to achieve the described quality criteria. Below, the features used are described in detail. Note that the actual values used are based on preliminary testing and estimates how undesired the respective situations are, and have shown to give very good results in general. Naturally they can be modified easily to enforce some aspects more than others, making this approach very flexible.

3.2.1. Coverage

Five different coverage types are distinguished:

1. completely uncovered: 1500 penalty points
2. covered only once, but enclosed between two non-letter-fields (either horizontally or vertically): 75 penalty points
3. covered only once: 200 penalty points
4. covered more than once in the same direction: 600 penalty points
5. covered once horizontally and vertically: 0 penalty points

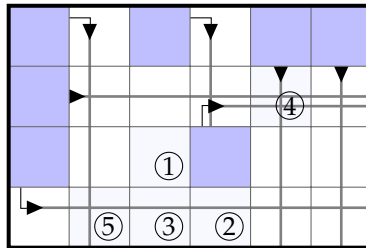


Figure 3.2.: Different types of field coverage. For each type, one example is highlighted.

Type 2 often is forced to occur, for example at the left or top border of the mask - hence it is penalized less than the similar type 3. Also note that type 4 is penalized much less than type 1, although it is an equivalent violation of the validity constraints. The reason for this is fairly straight forward: while leaving one field uncovered might greatly benefit the rating of the surrounding fields, double covered fields either occur in groups of at least two, or a word of length one is involved - in either case the surroundings account for further penalty points.

3.2.2. Word Lengths

Word lengths are rated in two ways. Mainly penalty points are given for the length of each word, according to a predefined function shown in table 3.1.

Word Length	Penalty Points	Word Length	Penalty Points
0	1800	8	50
1	1500	9	150
2	650	10	250
3	100	11	400
4	10	12	550
5	0	13	750
6	0	14	1000
7	30	15	1300

Table 3.1.: Penalty points for different word lengths

In addition to that, intersections of two words both longer than six letters receive additional penalty points given by the product of the lengths of the two words (i.e. a field where, for example, two words with length 9 intersect receives an additional 81 Penalty Points). This accounts for the fact that finding fitting words in such situations is especially hard.

3.2.3. Clustering

The size of each 8-connected (i.e. diagonal adjacency is considered as well as horizontal or vertical adjacency) cluster of definition fields is determined and penalized. As long chains of adjacent definition fields are especially undesirable, the maximum of the clusters horizontal and vertical extension is used as additional criterion for rating the cluster. An extract from the rating table is shown in table 3.2. It is to mention that definition fields at the left or top border of the mask are only counted half, as here clusters of size three or four are hardly avoidable and even desired.

Cluster Size	Maximal Extension	Penalty Points	Cluster Size	Maximal Extension	Penalty Points
1	1	0	5	3	794
2	2	150	5	4	882
3	2	288	5	5	980
3	3	320	6	4	1053
4	2	542	6	6	1300
4	3	603	7	5	1620
4	4	670	7	7	2000

Table 3.2.: Penalty points for different cluster sizes and extensions

3.2.4. Invalid Definition Fields of Type 3,4,5,6

The validity constraint for every word to be enclosed in between two non-letter fields is, in contrary to the other three constraints, not yet represented. Hence a penalty of 2000 is introduced for each word violating this constraint.

3.2.5. Dead Ends

Dead ends are, as defined in chapter 2.3, letter fields enclosed by three adjacent non-letter fields, excluding the fields at the top- or left border. These are penalized with 400 points.

3.2.6. Result

It is to note that the more fields a mask contains, the higher the best achievable score will be - simply due to the fact that not all of the above points can be avoided completely. In fact several of the above criteria work against each other: A perfect clustering score for example can simply be achieved by not using any definition fields at all - but then the coverage score would be extremely bad.

Figure 3.3 shows a plot of the rating of the best individual over a typical run of one thousand generations, split up into quality and validity parts. It can be seen that, while the validity component dominates at first, it is reduced very quickly, such that approximately from generation 100 onwards, the best result is always a valid mask. Also note that within the first one hundred generations, the validity and quality components frequently increase temporarily, while the total score is always decreasing.

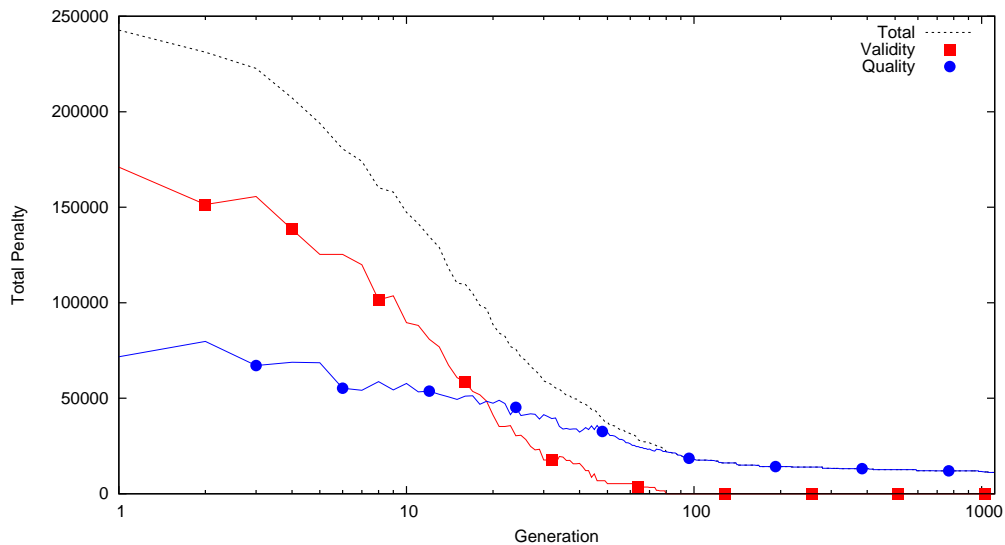


Figure 3.3.: Rating of the best individual over a run of one thousand generations. Note the logarithmic scale on the x-axis

3.2.7. Localized Fitness

One advantage of the presented fitness function is, that a rating for each distinct field of a mask can be approximated. This is done by, for example, distributing the penalty points a large cluster receives among all fields contained in that cluster. This allows to estimate the rating of only one half of a mask, or to localize areas that are especially “bad”, and hence need to be improved. This concept will be used in the second part of this work. A short

discussion about the *practical* value of the masks generated with this fitness function can be found in 5. Some (larger) exemplary, automatically generated masks can be found in appendix B.

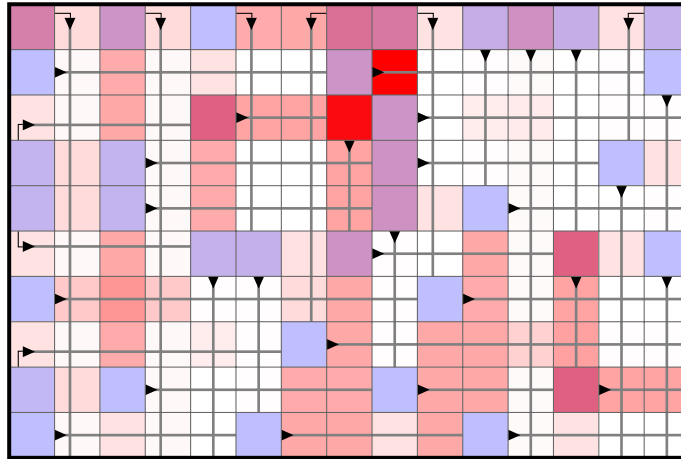


Figure 3.4.: The higher the local penalty rating of each field, the more red it is depicted.

3.3. Initialization

In a traditional genetic algorithm as explained in chapter 1, no problem domain knowledge is used apart from the fitness function and the way a solution is encoded. Still using such knowledge - if available - can significantly improve the results, as one can “guide” the search towards the assumed position of an optimal solution. This however has to be done with great care, as by imposing additional constraints one can accidentally modify the search space in such a way, that finding a (globally) optimal solution becomes far more difficult.

Despite these concerns, two additional modifications were made. As a first and obvious step, field assignments which - no matter how the surrounding mask looks like - are certain to cause a validity violation are disallowed.

In order to justify the second - far more restrictive - modification, we first need to examine an often occurring phenomenon, shown below. Avoiding this kind of arrangement by including a penalty in the fitness function proved to be fairly ineffective: this can be explained by the fact that several fields have to be changed in order to resolve such a situation without creating new violations - hence making it a good example for a local minimum.

The solution was to completely remove these arrangements from the search space by

- disallowing most of the definition field assignments creating such situations
- instead of using completely random initial individuals, preassigning the border-fields with (random) fitting field types, using a (fairly simple) static algorithm.

It also showed that initializing the rest of the mask with letter fields only improves the performance even further. One examples for a resulting initial individual can be found in

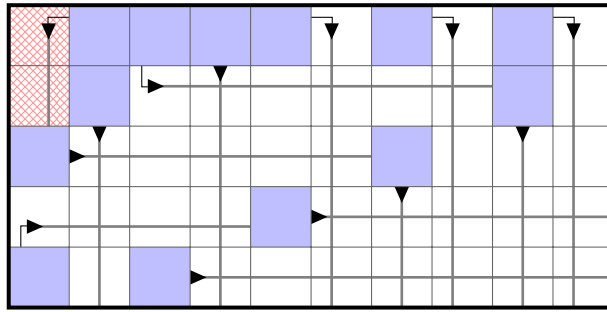


Figure 3.5.: Words directly adjacent to a parallel border are to be avoided, such a situation is highlighted in red. However in order to resolve this particular situation without introducing at least one new validity violation - resulting in an overall worse penalty (note that the mask as it is does satisfy all validity constraints) - four fields would have to be changed simultaneously - which is (especially for bigger masks) *very* unlikely to be achieved by a mutation.

figure 3.7. Note that *all* fields are still subject to mutation and crossover, hence the border fields are allowed to be changed during the algorithm as well.

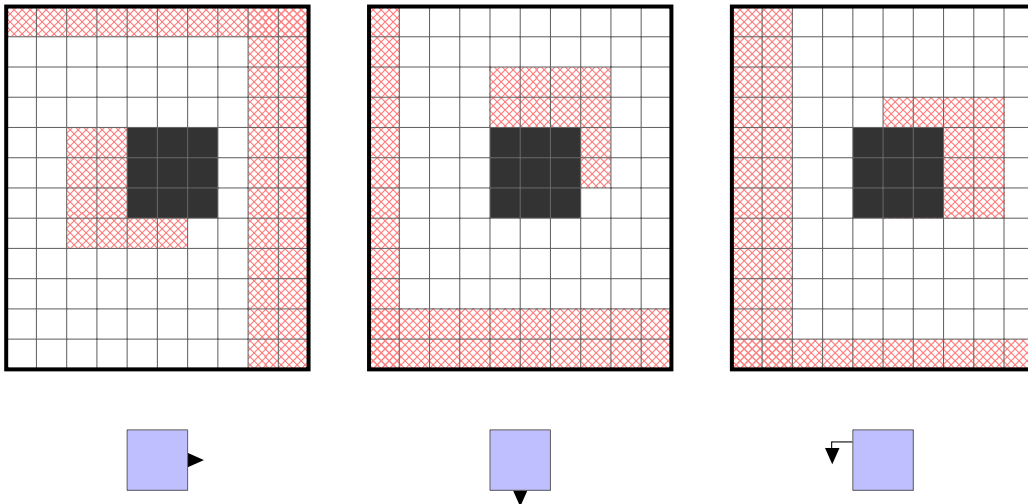


Figure 3.6.: Examples for some disallowed field assignments. The field type shown below each mask is not allowed at the highlighted fields.

3.4. Mutation

A fairly obvious way to implement a mutation operator is to simply replace k randomly chosen fields with new random field types (subject to the constraints discussed in the previous part). In order to improve the effectiveness of this operator, several modifications were made:

- **Field Type Probabilities:**
When choosing a new random field type, no uniform probability distribution is used.

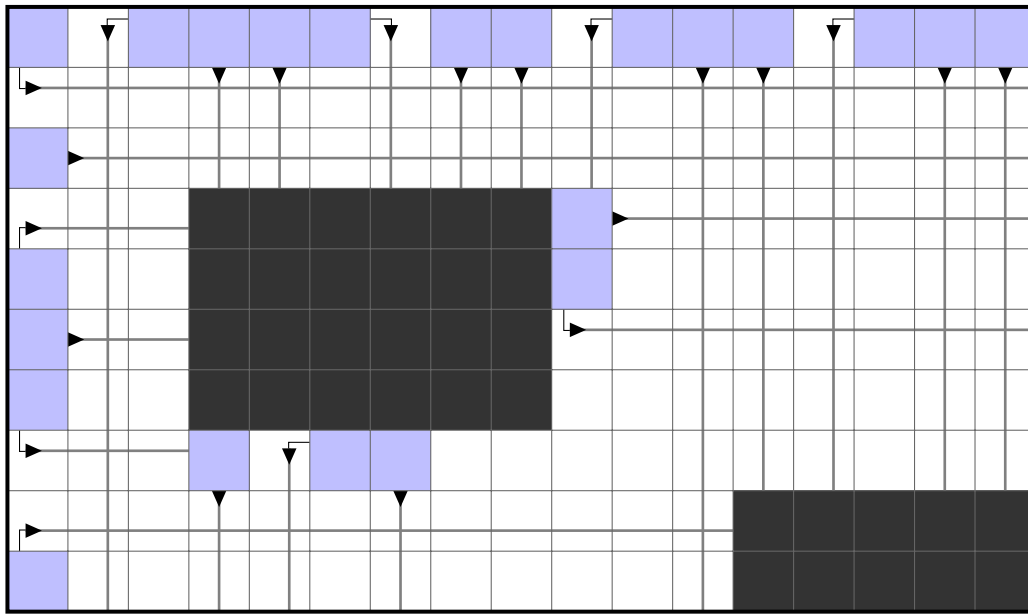


Figure 3.7.: An example for an initial individual. Not only the fields adjacent to the left / top border are assigned, but also fields adjacent to such borders created by cut-out fields (which are given from the beginning).

As a typical mask contains approximately $\frac{2}{3}$ letter fields, a letter field is chosen with probability $\frac{2}{3}$. Furthermore, definition fields of type 1 and 2 occur far more often than type 3, 4, 5 and 6 - hence of the remaining $\frac{1}{3}$, type 1 and 2 each get a probability of $\frac{1}{12}$ and type 3, 4, 5 and 6 each $\frac{1}{24}$.

- **Centralized Mutation:**

Instead of determining k fields to be modified separately, the first field, the *central point* is chosen using a uniform distribution. The remaining fields to be modified are then chosen close to this central point¹, resulting in a more local mutation. The reason for this is the following: Changing two or more adjacent fields often helps overcoming a local optimum. At a bigger distance however, two changes are unlikely to be correlated at all - together with the fact that a random change is, at least in the later phase of the evolution, far more likely to have a (significant) negative effect than to achieve an improvement, the probability for two uncorrelated changes together to still improve the total score is extremely low.

- **Guided Mutation:**

The probability for the central point to be chosen in areas with a high number of penalty points was increased slightly. This was done using the principal of tournament selection: α fields are chosen at random, and the one with the highest local penalty score is selected as central point. In practice, $\alpha = 2$ gave the best results; higher values again leading to premature convergence.

¹To be exact, a two-dimensional normal distribution with $\mu := (x, y)^T$ and $\Sigma := \sigma^2 I_{2 \times 2}$, with $(x, y)^T$ denoting the central point was used. Preliminary testing showed that $\sigma = 3$ is a good choice for the standard deviation, smaller values leading to premature convergence.

• **Predefined Mutation:**

Two fixed mutation types, occurring with a certain probability, were introduced:

- “shifting” a definition field of type 1 or 2 one field horizontally or vertically.
- “splitting” a long word in two halves, by inserting a field of type 1 or 2.

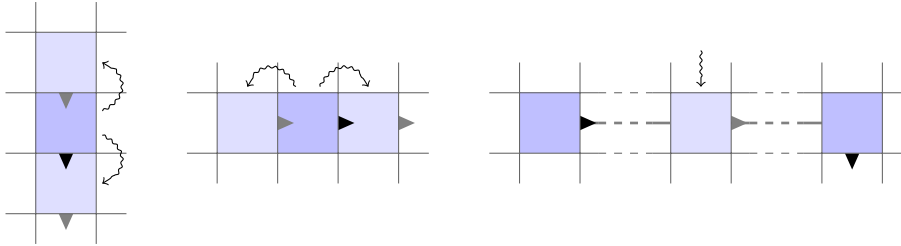


Figure 3.8.: “shifting” and “splitting”.

In order to compare the effectiveness of this operator for different values of k , a simple hillclimber was used. For each setting, thirty 20×20 masks were created. The continuous line depicts the average of those thirty runs, the boxes depict the average plus / minus the standard deviation. The whiskers depict the best / worst result respectively. Pseudocode of the basic mutation operator can be found in appendix A.

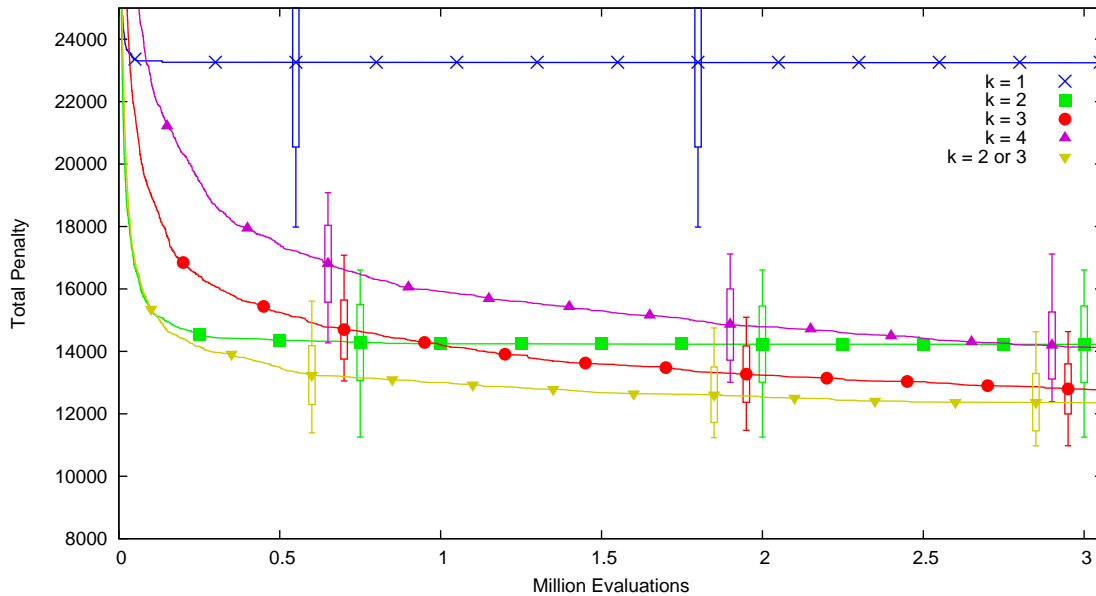


Figure 3.9.: Results for different values of k . The best result was achieved if k is chosen at random from $\{2, 3\}$.

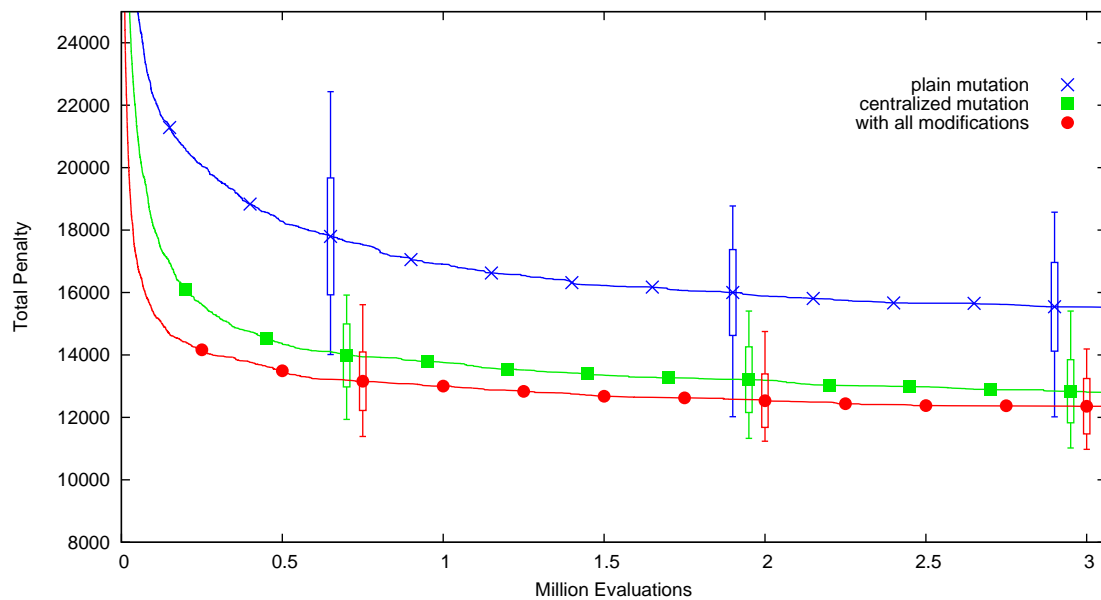


Figure 3.10.: Results when using none of the modifications explained above, only centralized mutation or all modifications; all runs were executed with k randomly chosen from $\{2, 3\}$. It can be observed that using a centralized mutation has a huge impact on the effectiveness of the mutation operator, while the other three modifications mainly increase convergence speed and only lead to a slightly better overall result.

3.5. Crossover

In order to better understand the results presented in this chapter, we first need to highlight one very distinctive property of the considered problem.

As discussed in the first chapter, for all relevant optimization problems there exist (more or less strong) dependencies between the parameters to be found. For the problem of generating crossword masks, fields close to each other are extremely dependent while fields more than four cells apart hardly have any (direct) influence on each other at all, as can be observed in 3.11. In fact, whether a specific field assignment is beneficial or not in most cases solely depends on the assignments to the surrounding fields.

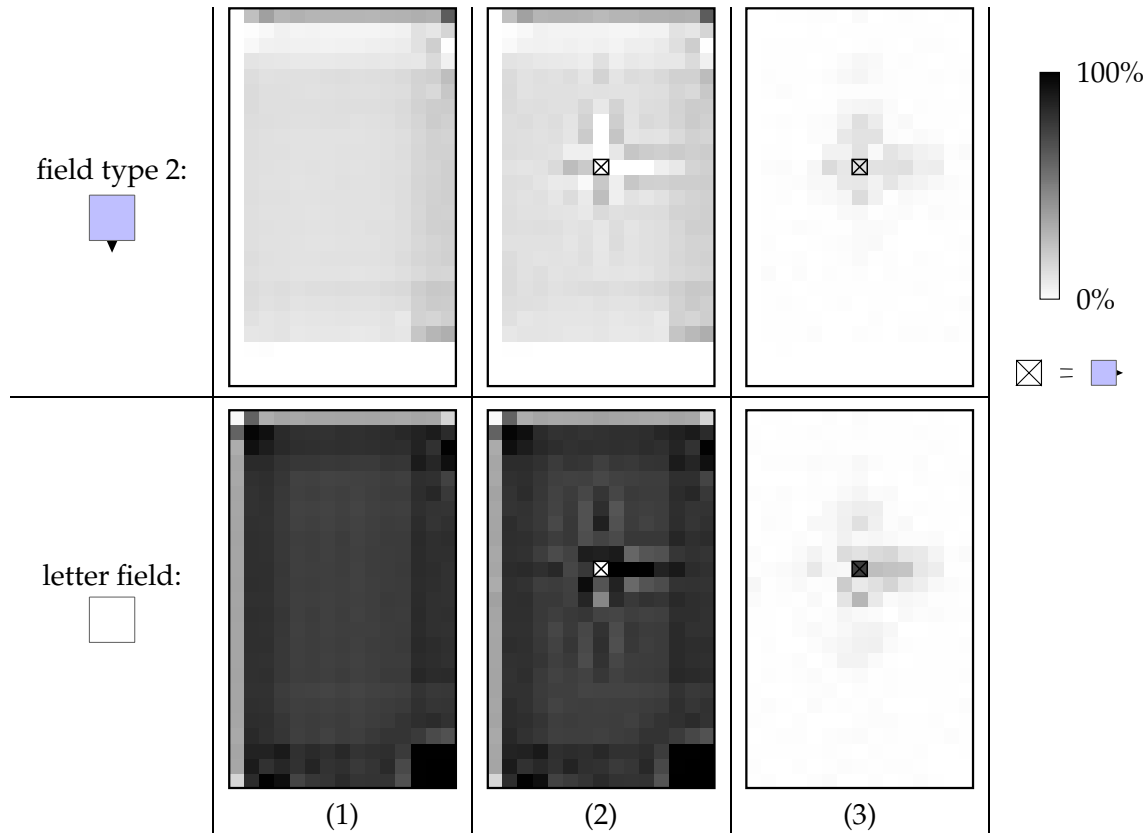


Figure 3.11.: For this graphic, 50,000 valid masks were created. (1) visualizes for each grid cell the percentage of masks having a letter field or definition field of type 1 respectively at that position. (2) shows the result when only counting masks that have a definition field of type 1 (i.e. a straight arrow to the right) at the position marked with a cross. (3) shows the difference between (1) and (2). It can be observed that in (1) most fields (apart from those close to a border) are equally likely to be assigned a certain field type. When fixing one field however, this changes significantly for the cells close by, whereas cells more than four fields apart are hardly influenced at all - as can be seen in (2) and (3)

Following this consideration it becomes obvious that when fitting two halves of two different masks together, some of the field assignments close to the splitting line will loose

a lot of their value, as they were only a good choice in combination with the - now replaced - other half of the original mask. It is therefore not surprising, that the shorter the splitting line is, the better the resulting masks are. As a very short splitting line can obviously be achieved by, for example, just cutting off one of the corner fields, an additional criterion - for example that each half has to contain at least 30% of the overall non-cut-out fields - is required. For simplicity reasons only lines through the central point are allowed². Pseudocode of the resulting crossover operator can be found in appendix A.





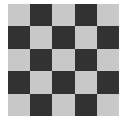

Mask:						
Average Result:	30,914	37,399	46,623	74,411	129,024	227,873
Success:	24	2	0	0	0	0

Figure 3.12.: Different crossover masks, tried on every possible combination of 500 masks (i.e. 249,500 tries). Average score of original masks: 17,207. success denotes how often the resulting mask was better than *both* respective parents.

The result of such a crossover operator however is fairly discouraging: especially for big masks, the result of crossing two completely different masks is *very* unlikely to actually be better than the parents, as can be seen in 3.12. This is due to the strong dependencies between adjacent fields and the consequence that new validity violations produced along the crossover line outweighs any positive effects a crossover might have. Two additional things have to be added:

- The results presented in 3.12 were obtained using masks that already are “fairly good”, meaning they contain at most two validity violations. When using less evolved masks, the results are similar but less extreme: some newly produced validity violations have far more impact on the fitness value if the original masks only contain very few violations. However as a mask only containing very few to no validity violations at all is achieved quite quickly, the main challenge lies in optimizing such a mask with respect to the quality criteria - justifying these considerations.
- The more alike the two parents are, the better the results of the crossover operator will be - simply due to the fact that less dependencies are destroyed. This leads to the following consequence: If the population has a high diversity - which usually is desired - the crossover operator is too destructive to be used efficiently. If the population however has a very low diversity, the crossover operator is far less destructive, but simultaneously the main advantage of a genetic algorithm - the ability to explore the search space efficiently and exhaustively - is lost (see chapter 1.5, “Diversity vs. Convergence”).

²Sampling splitting lines with respect to the proposed 30%-criterion is not trivial, as arbitrary cut-out field positions have to be taken into account. It however could be done - apart from a simple *try & error* approach - in adequate, constant time (using appropriate pre-computation).

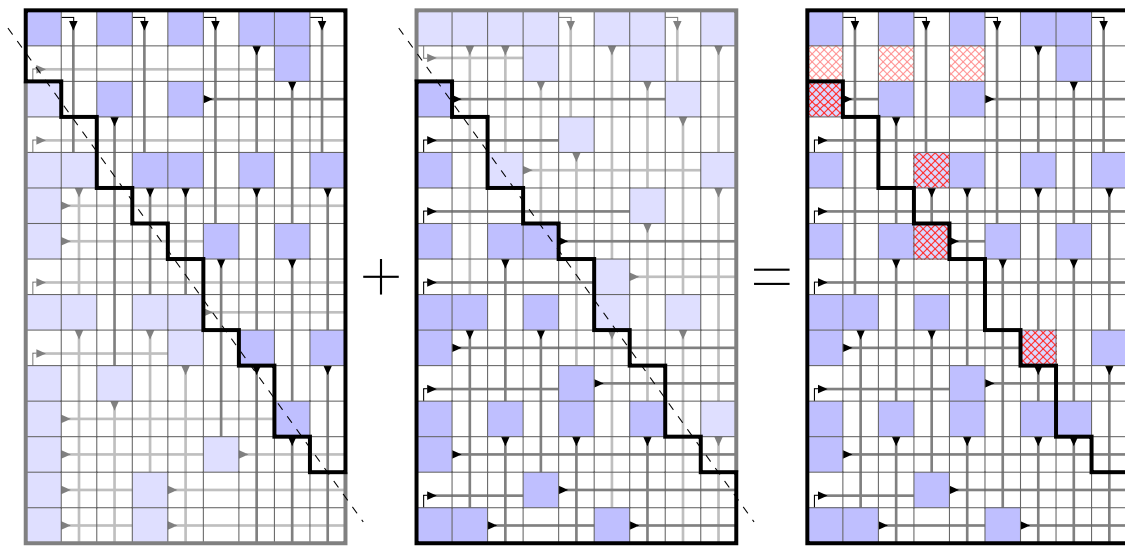


Figure 3.13.: An example for crossover. While both parent masks meet all validity constraints and achieve a total penalty score of 5,920 and 5,490 respectively, the mask resulting from a crossover along the dashed line contains seven violations and hence has a far worse total penalty of 15,925.

3.6. Results

Having defined the fitness function as well as the mutation and crossover operator, there are still some parameters left that can be adjusted:

- **Selection Function:** Throughout this work the so-called tournament-selection presented in the first chapter was used. This leaves the parameter $\alpha \in \mathbb{N}$ (determining the tournament size) to adjust the selection pressure. Default value: $\alpha = 4$.
- **Population Size** =: $n \in \mathbb{N}$ denoting the number of masks in each generation. Default value: $n = 200$.
- **Crossover Rate** =: $c \in [0, 1]$ denoting the fraction of the new generation to be created by crossover. It was decided to use an elitist selection, meaning that the best mask from the parent generation is always copied to the next generation without being changed. The fraction of the new generation created by mutation therefore is approximately $1 - c$. Default value: $c = 0.5$.

The following plots document the results achieved with different settings for these three parameters. For each set of parameters, thirty evolutions were run, generating a 20×20 mask without any cut-out fields. The continuous line depicts the average over all thirty runs, the boxes denote the average plus / minus the estimated standard deviation. The best and worst results are depicted by the whiskers of the box plots. Pseudocode for the genetic algorithm, as well as for a simple hillclimber, can be found in appendix A.

Note again that due to the design of the fitness function, the actual rating achievable is highly dependent on the mask dimensions and the positions of cut-out fields. In particular a mask with rating 0 is not achievable: in fact the overall best 20×20 mask found still had a rating of 8,047.

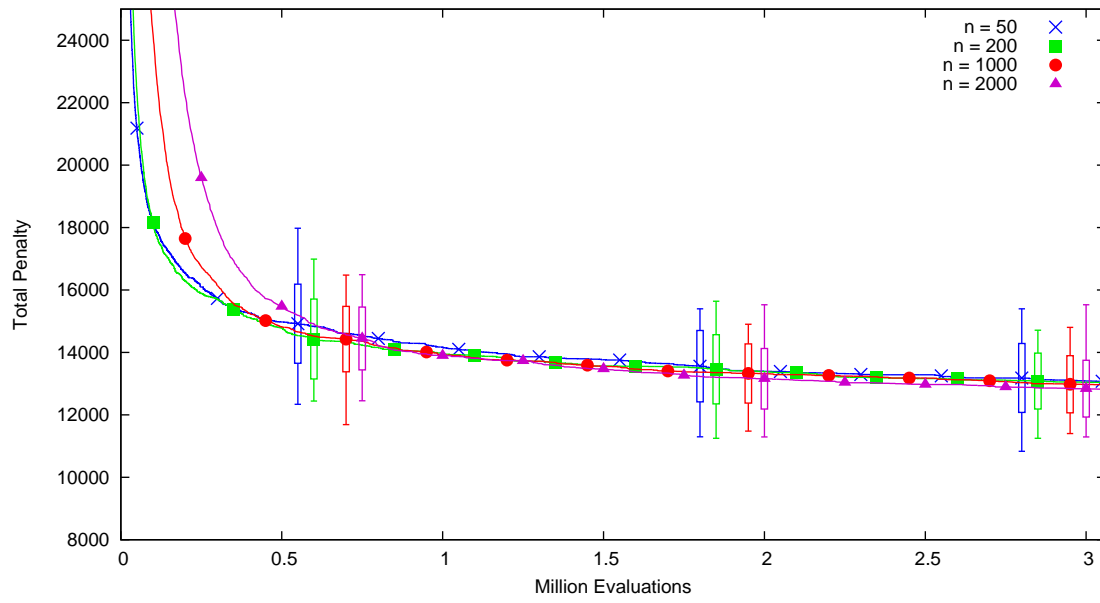


Figure 3.14.: Results for different values of n (population size). $\alpha = 4, c = 0.5$.

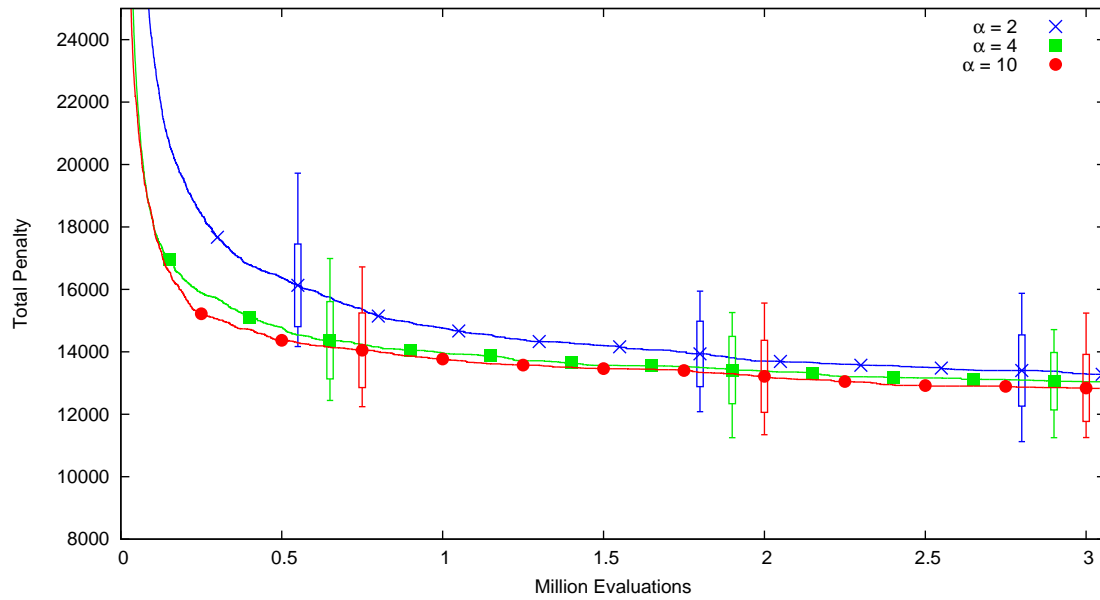


Figure 3.15.: Results for different values of α (tournament size). $n = 200, c = 0.5$.

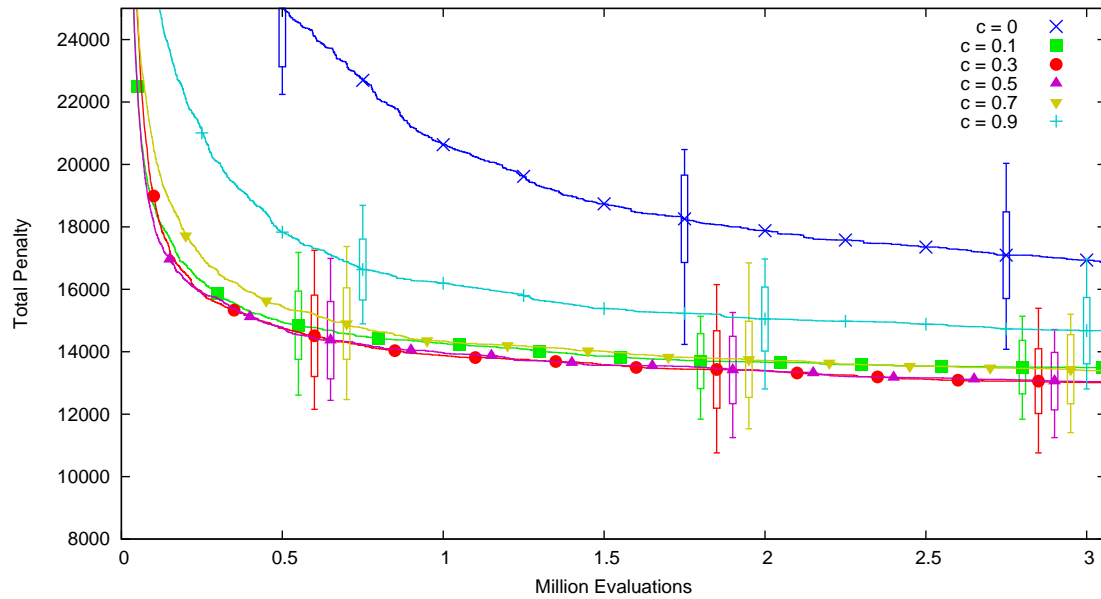


Figure 3.16.: Results for different values of c (fraction of new generation created by crossover). $n = 200$, $\alpha = 4$.

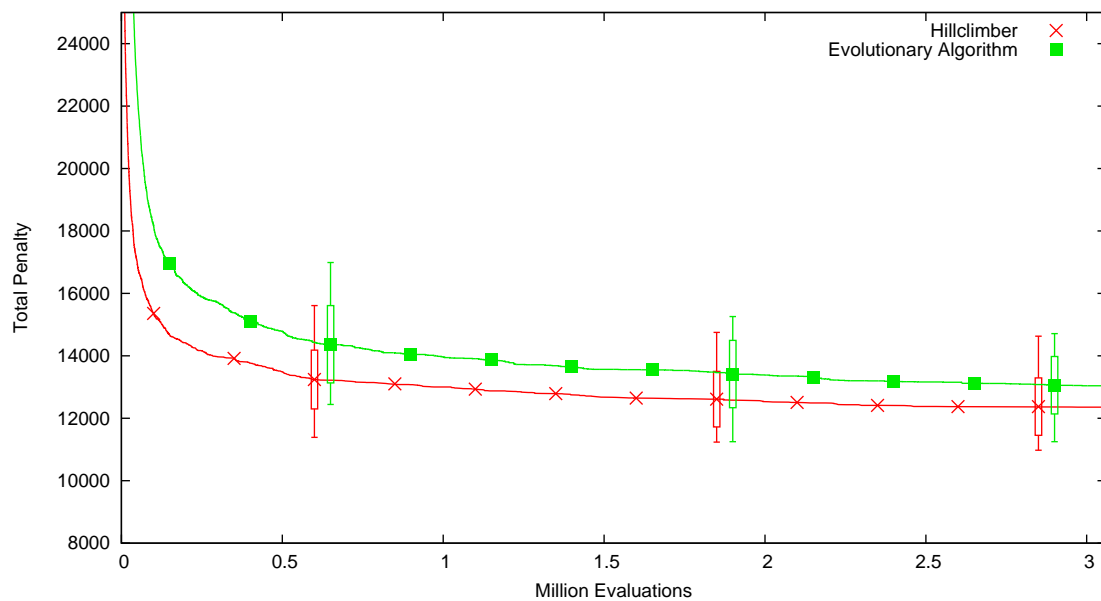


Figure 3.17.: Result of a simple hillclimber compared to the evolution with default settings ($n = 200$, $\alpha = 4$, $c = 0.5$).

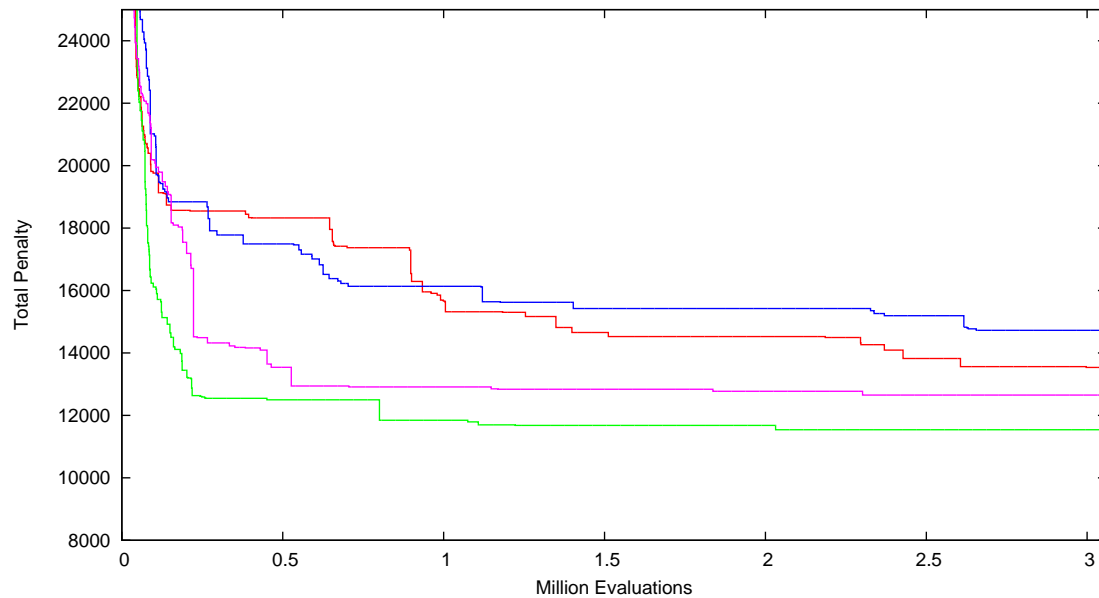


Figure 3.18.: Results for four exemplary single runs with default settings ($n = 200$, $\alpha = 4$, $c = 0.5$). It is to note that rather than decreasing continuously, relatively few but significant improvements are found which, when averaged, result in the smooth graphs found in the plots before. This is due to the discrete nature and the high dimensionality of the problem, leading to a huge number of possible mutations with only a fraction being beneficial.

3.7. Discussion

From these plots it can be concluded that - for this specific problem - the advantages an evolutionary algorithm usually has compared to a simple hillclimber can not be exploited: **A simple hillclimber performs slightly better than a complete evolutionary algorithm.** Further examination shows that this is mainly due to two points:

- **Extremely quick Convergence:** It turns out that even for very low selection pressure ($\alpha = 2$), the diversity among the population decreases very quickly. This can be observed in the below plot, showing the average hamming distance (i.e. number of differently assigned fields) between all masks within each generation. Even omitting the elitist selection, i.e. not directly copying the best individual, leads to no significant difference. In fact it is questionable whether a significantly higher diversity was beneficial at all - as the crossover operator would, as discussed in the previous chapter, hardly be able to exploit this. Hence no further actions aimed at increasing the diversity were taken.

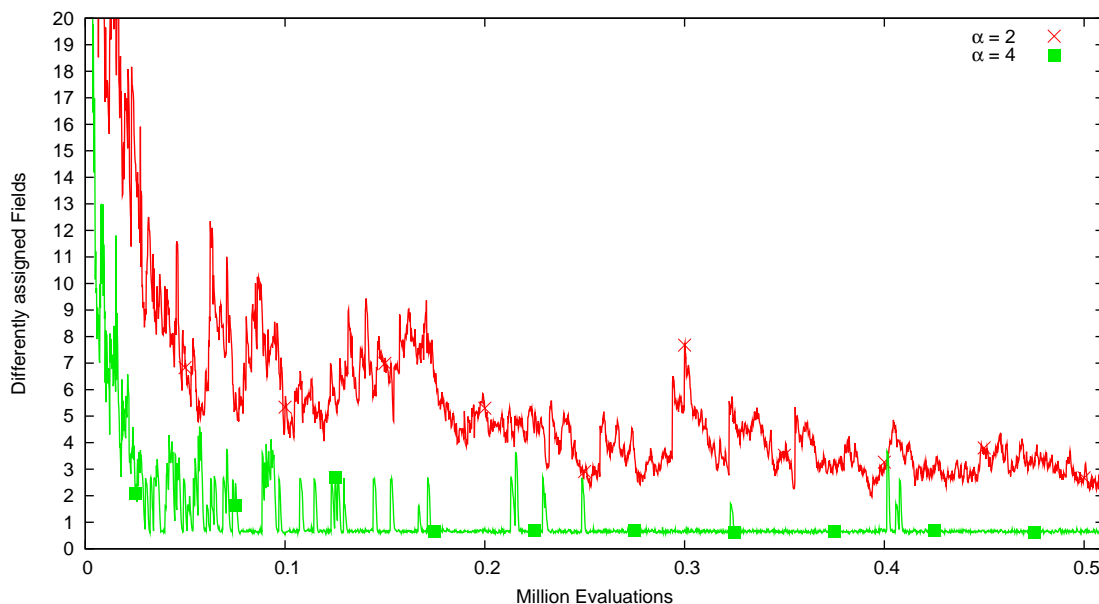


Figure 3.19.: Average Hamming distance within each generation for a 20×20 mask and $n = 200$, $c = 0.5$ and $\alpha = 2$ or 4 respectively; for the first 500,000 evaluations (corresponding to 2,500 generations). Note that the average actually drops below one, meaning that the same mask (probably the currently best one) occurs multiple times within the population, as crossover “rediscovered” the original mask by patching together the unchanged halves of two mutation offsprings.

This development is responsible for the observable result that increasing α above 4 has no visible effect: As especially the better individuals are extremely similar or even equal, it does not make a significant difference whether the best or the tenth best individual is selected. It also accounts for the fact that higher values for n do not lead to (significantly) better results in the long run: the increased capability to explore the search space simply is not exploited. It does however take longer to converge, as in the beginning a lot of evaluations are wasted on masks that get thrown out anyway.

- **Low Mutation Success Rate:** Due to the discrete nature of the problem and very high dimension, there is a huge number of possible mutations. For a 20×20 mask for example, there are approximately two billions of possible mutations of size three. When only considering mutations where the changed fields are at most six fields apart (which is reasonable for the centralized mutation used), there are still in the order of 25 million possibilities left. In the later part of the evolution, the vast majority of these mutations result in some new validity violation or a significantly worse quality rating, making the resulting mask worse than the original mask - although there still are some improvements possible. This can indirectly be observed when looking at the fitness plot for a single evolution (see figure 3.18): The fitness value of the best mask stays constant for - in the later part - thousands of generations, until at some point an improvement is discovered, leading to a significantly better value. The below graph shows the empiric mutation success rate for a hillclimber run on a 20×20 mask. Each cross represents a successful mutation.

In this scenario, “wasting” mutations on suboptimal masks might not be that beneficial. Still it turns out that crossover is required for extracting improvements found by mutation - this is necessary, as the mask on which an improvement was found might have had some flaw compared to the previously best mask itself (for example resulting from some unsuccessful mutation). Crossover then is able to extract the found improvement and “repair” this flaw. Hence, the evolution performs significantly worse if no crossover is used at all ($c = 0$, see figure 3.16).

With the current implementation, about eight thousand 20×20 masks can be evaluated per second on an average home PC. A run over three million evaluation hence takes approximately six to seven minutes - which is practically feasible

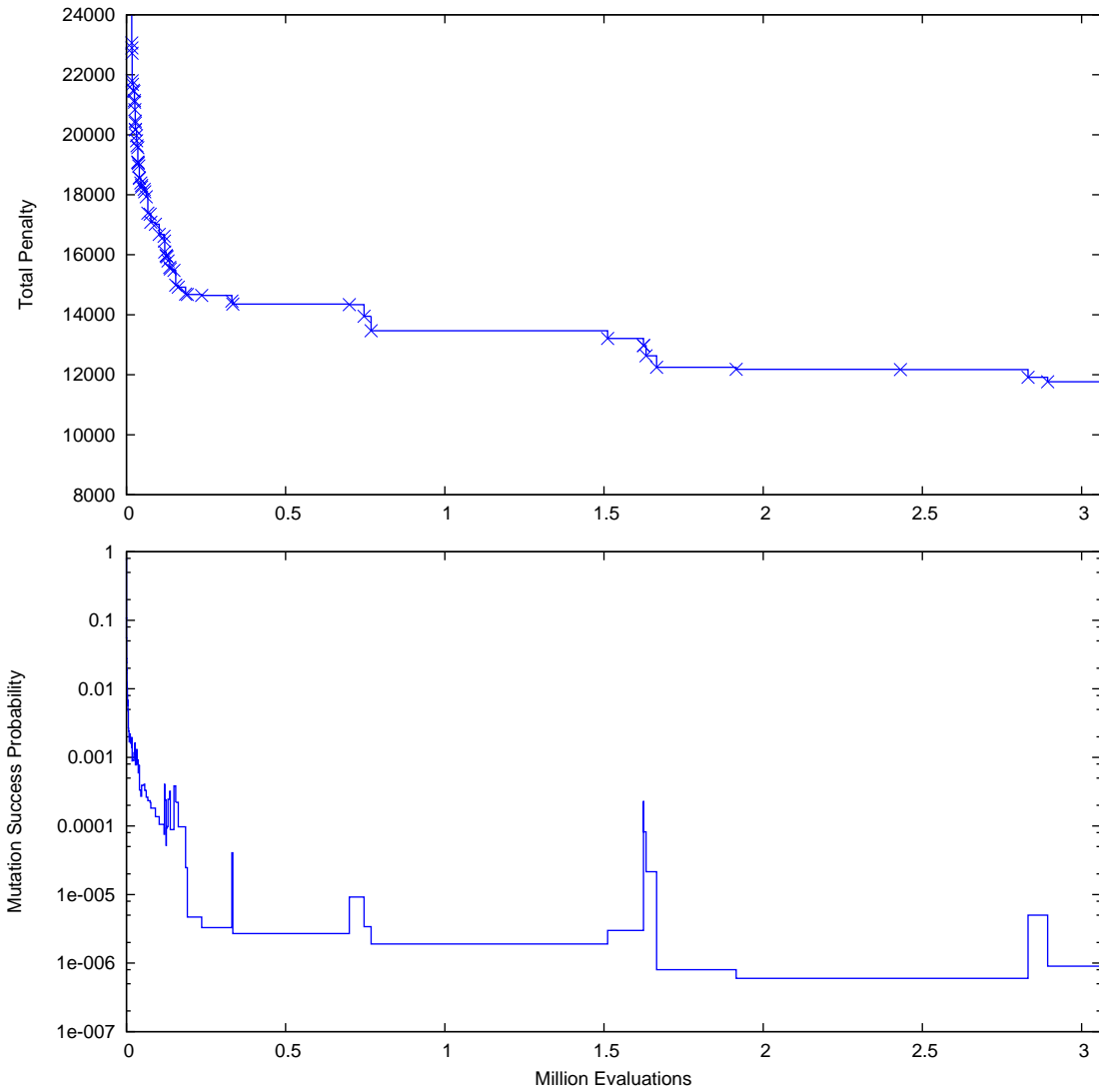


Figure 3.20.: Mutation success rate and the corresponding rating of an exemplary hillclimber run. Note the logarithmic scale in the below plot.

4. Memetic Algorithm Approach

4.1. Basic Idea

The results discussed in the previous chapter - a simple hillclimber exceeding a complete evolutionary algorithm - are primarily due to the extremely strong *local* dependencies, and the resulting poor performance of the crossover operator. Apart from the problems arising along the splitting line however, a crossover can in fact be very beneficial, as globally different areas of the mask hardly influence each other's value at all.

It turns out that in most cases, the validity violations (and of course assignments contributing to a bad quality rating) created by a crossover along the splitting line can easily be corrected by applying a hillclimber to the resulting child in order to "repair" it. Consider the mask resulting from the exemplary crossover in chapter 3.5: only four independent mutations, each of size one (and hence relatively likely to be discovered), are required to repair the mask - making the result significantly better than both parents.

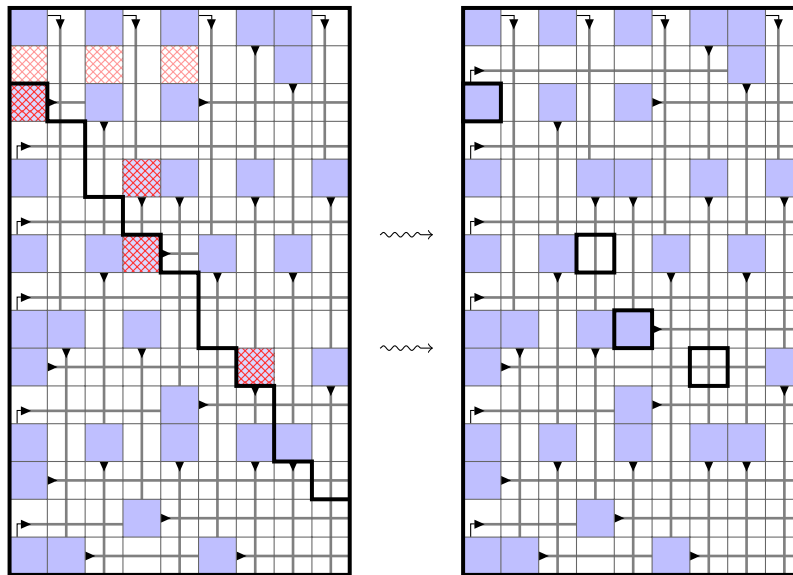


Figure 4.1.: The original crossover result depicted on the left contains several validity violations, and has a rating of 15,925. With only four independent changes however, the mask can be "repaired", such that it achieves a rating of 4,890 - better than both parents (which had a rating of 5,920 and 5,490, see figure 3.13).

This gives rise to the concept of introducing a "second" evolutionary algorithm on a higher level: this evolutionary algorithm only uses crossover, but a hillclimber is applied after each crossover to repair the resulting child mask. This concept is also known as **Memetic Algorithm** [?]: Basically exploration and exploitation are separated: While ex-

ploration is mainly done by the “outer” evolutionary algorithm using only crossover, the hillclimber process is responsible for exploitation - which is what a hillclimber is best at.

4.2. Implementation

This approach however can still be refined: The potential of a mask resulting from a crossover can be estimated *before* applying the hillclimber. This is done by using the local rating of both parents: Let the **potential rating** of a crossover result be defined as the *sum of the local ratings of both halves*. This value only accounts for the quality of both halves on their own, without taking the problematics arising along the splitting line into account - assuming that these are “repaired” by the hillclimber, this is a reasonable choice¹.

Using this value, a preselection of crossover results based on their potential rating can be determined, and only this preselection is repaired with a hillclimber, as the hillclimber is the - by several orders of magnitude - computationally most expensive step.

Still there are some masks, which turn out to be difficult to repair (due to newly created local minima), or which - even after being repaired - just are not that good and hence are not worth to be exploited fully. These are filtered out by first applying a hillclimber with a relatively weak break condition to all masks, then selecting only a subset having the best rating, and then continuing the hillclimbing process only on this subset. At this point it also is sensible to try to keep the diversity within the population as high as possible, as the genetic algorithm basically is only responsible for exploration anyway. In figure 4.2, the basic schematics of the resulting algorithm are depicted; additionally it can be found in appendix A.

¹As one could expect, this value still is significantly lower than the actual rating of a mask after “repairing” it.

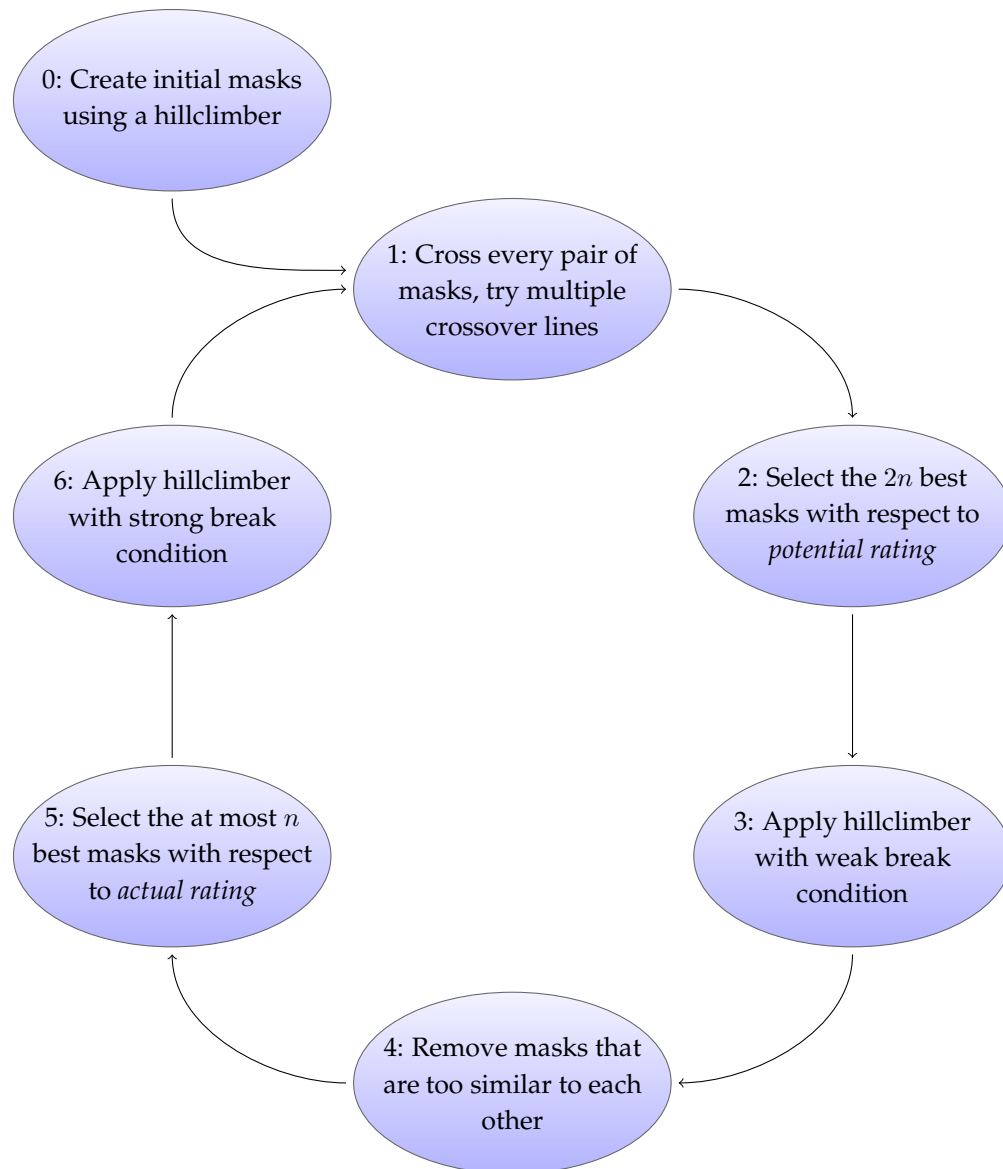


Figure 4.2.: General layout of the algorithm. n stands for the population size. Note that *two* selection steps are present, both of which are completely deterministic - i.e. always the best masks are selected. The - by several orders of magnitude - most expensive steps are the two hillclimbing processes applied to all masks, i.e. step three and six.

4.3. Results

In this section, the results obtained when using the algorithm described above are presented. Note that for the following plots only eight runs were done - each run however was allowed twenty million evaluations (instead of only three million, as in the previous chapter). The first plot directly compares the performance of a simple hillclimber with the performance of the memetic approach. Afterwards different values for the following four parameters are tested:

- δ := maximal fraction of identically assigned fields, for two masks to be considered as “too similar”. Default value: $\delta = 0.1$
- n := population size. Default value: $n = 15$
- b_w := weak break condition: Break after b_w unsuccessful mutations in a row. Default value: $b_w = 2,000$
- b_s := strong break condition: Break after b_s unsuccessful mutations in a row. Default value: $b_s = 10,000$

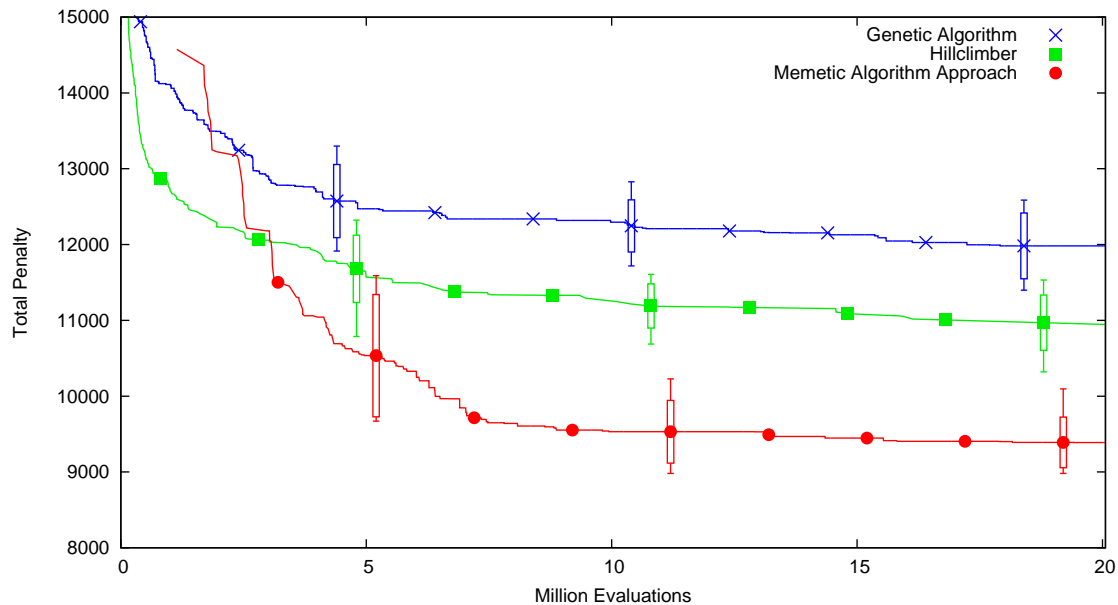


Figure 4.3.: Memetic approach vs. basic hillclimber.

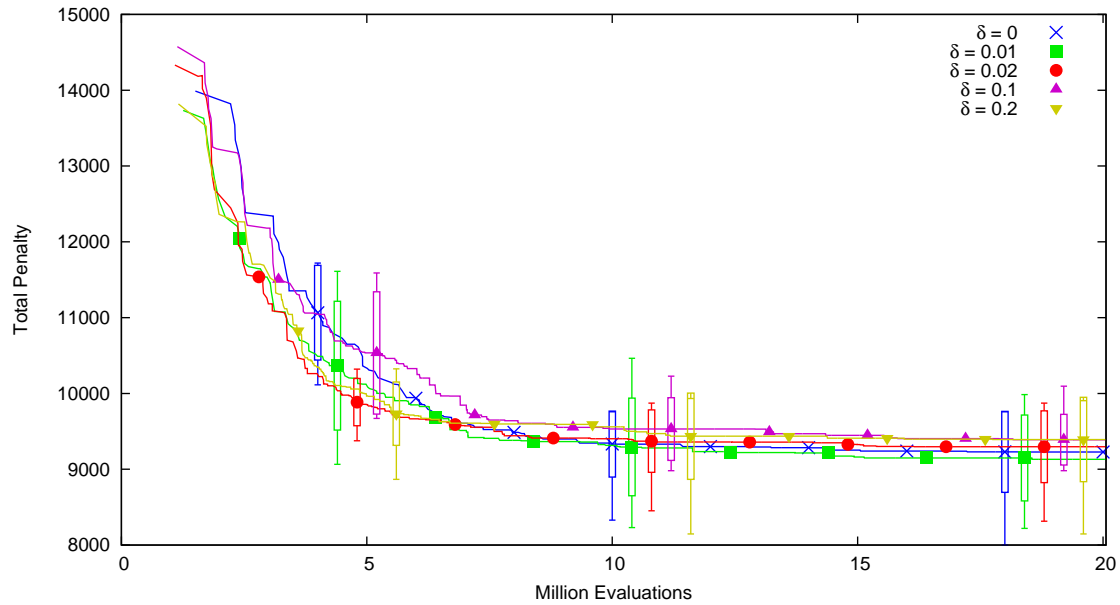


Figure 4.4.: Different values for δ : Some optimization might be possible here.

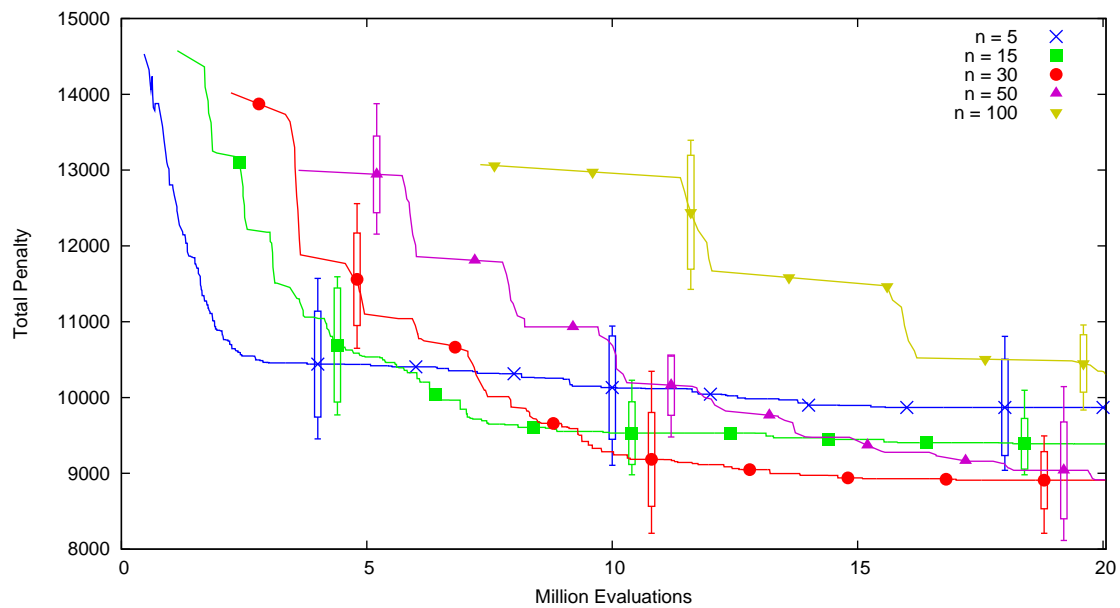


Figure 4.5.: Different values for n : These results are pretty much what one would expect: the bigger the population, the more exploration is involved - leading to slower convergence but (up to a certain point) overall better results

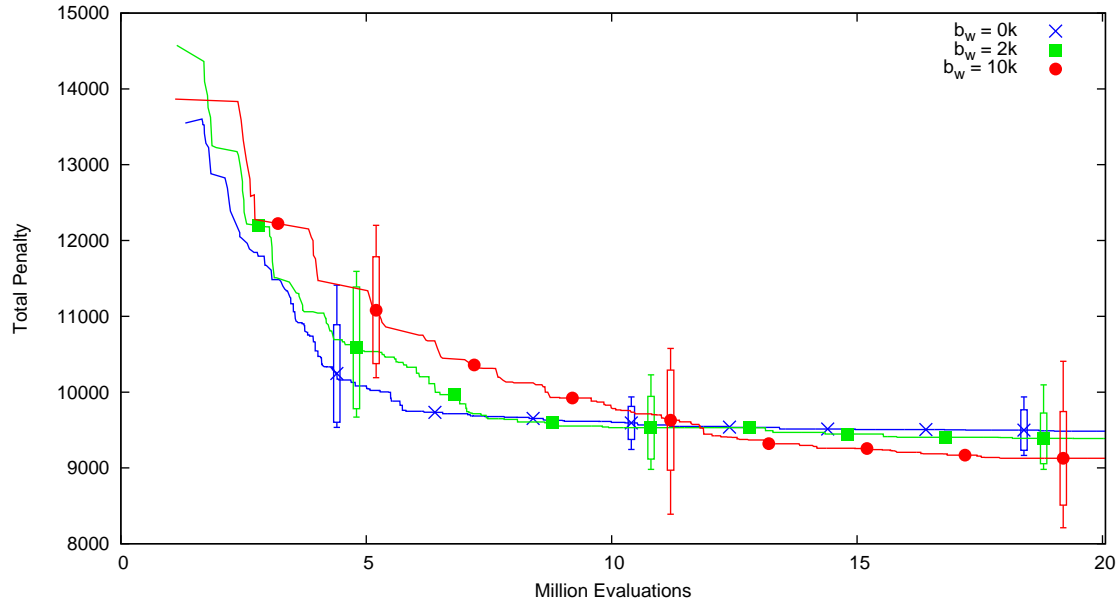


Figure 4.6.: Different values for b_w

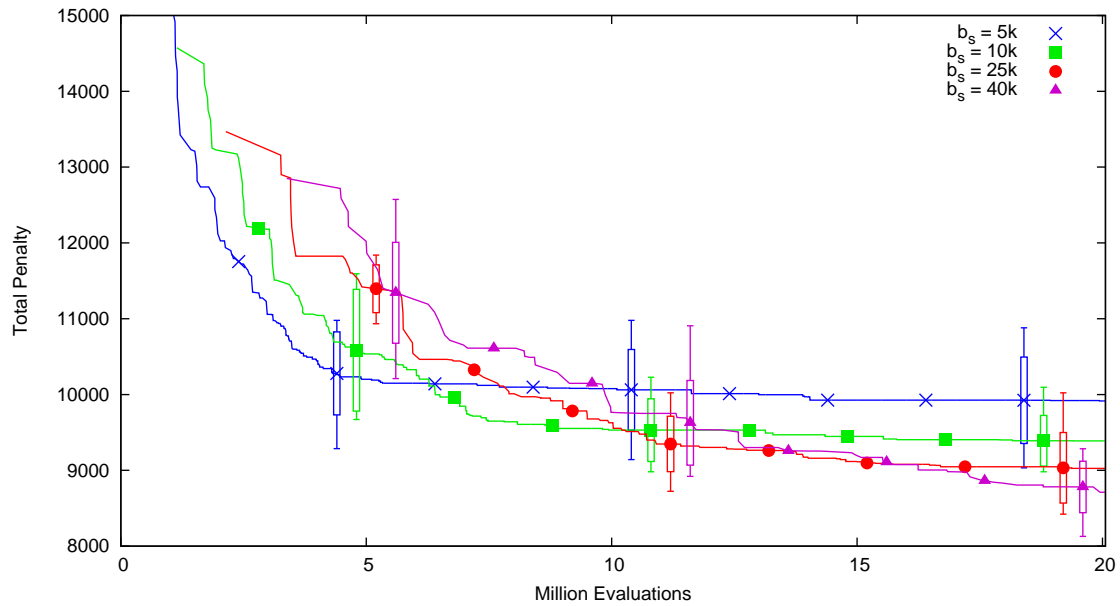


Figure 4.7.: Different values for b_s : Again, by adjusting b_w and b_s , better overall results can be achieved, but in turn more evaluations are required.

5. Practical Results

In the preceding chapters, different algorithms and parameters were judged based solely on the fitness values of the masks created. This makes sense, as the fitness value is the only information about a mask available for the algorithm - hence the better the fitness of the masks created, the better the algorithm. Whether the fitness function used really represents the desired characteristics for a good crossword mask is irrelevant for this process, as the performance of an algorithm can be assumed to be independent of the exact fitness function used¹.

From a practical point of view however, the resulting mask itself is relevant - and not some fitness value. The link between these two points of view is the fitness function. Judging a fitness function however is only possible by analyzing the resulting masks and - in order to improve it - one needs to adjust the different penalty values and identify new features to be penalized (or maybe rewarded). For this thesis, a professional opinion from Axel Ruepp Rätselservice² was solicited:

“[The masks] are surprisingly good, almost as good as handmade. Only very few adjustments are necessary to make them fit for being used in practice.”

Some exemplary resulting masks can be found in appendix B: Based on manually created masks (again, by Axel Ruepp Rätselservice), masks with the same layout were generated by the memetic algorithm approach. Note that a slightly modified fitness function, taking some more complex (and technical) features into account was used.

An interesting point to add is that, regardless of which fitness-function settings are used (i.e. how the different features are penalized), the automatically generated masks are - by a large margin - always better than the manually created “originals” (with respect to the fitness function used). This strongly suggests that, apart from optimizing the running time of the algorithm, significant improvement is only possible by finding some more accurate fitness function.

¹At least for roughly similar fitness functions, this is a reasonable assumption.

²<http://www.raetselservice.de>

6. Further Work

As discussed in the previous chapter, probably no significant improvements concerning the quality of masks generated *with respect to a given fitness function* is possible. No matter how the fitness function is parameterized, the memetic approach already generates masks which are - with respect to that specific fitness function - better than manually designed masks. Still some optimization with respect to running time and convergence speed is possible: For one thing the memetic approach itself has not been examined very extensively, for several design choices different solutions might prove to perform better. For another thing, the mask evaluation process can be sped up significantly: Currently the whole mask is re-evaluated after each mutation. However especially in the context of a simple hill-climber where the same mask is mutated thousands of times until finally an improvement is found, it should be possible to only calculate the impact of a single mutation without re-evaluating the whole mask¹. In particular, this would lead to an evaluation running in $\mathcal{O}(1)$ instead of - as it is the case with the current implementation - $\mathcal{O}(\text{number of fields})$.

The fitness function itself however is a different matter: the main problem is that ultimately the only way to directly compare two fitness functions is to have an expert judge the resulting masks. At least the features of a mask to be rated - for example to judge a cluster based on the number of definition fields it contains - have to be identified manually, and there obviously are many more possibilities than only the basic ones presented here. How exactly a specific feature then impacts the rating of a mask - for example *how many* penalty points to give for which cluster size, and how the different features are weighted - could then be determined by some more interesting methods: For example using a set of manually created “prototype masks” one could try to automatically adjust these fitness function parameters to give *better* fitness values for the prototypes, while simultaneously giving *worse* fitness values for generated masks - hence iteratively leading to masks more and more similar to the given prototypes (again, only with respect to the features rated at all).

¹This however requires quite a complex logic, as even a single change can have quite a lot of consequences - leading to very technical code of hardly any scientific interest. In particular, it might be necessary to build some more complex representation of the parent mask, allowing to quickly have access to the affected words, clusters and so on - but as mentioned, in the context of thousands of mutations being performed on the same parent mask, this probably pays off.

Appendix

A. Code

Algorithm 1: Mutation

Input: Parent *mask*, Parameter *mutationSize*

```
1  $(x, y) \leftarrow \text{getRandomField}()$ 
2  $result \leftarrow \text{mask.copy}()$ 
3  $result[x, y] \leftarrow \text{getRandomFieldType}()$ 
4 for  $i \leftarrow 2$  to  $mutationSize$  do
5   repeat
6      $(x', y') \leftarrow \text{sample from } \mathcal{N}((x, y), 3 \cdot I_{2 \times 2})$ 
7     until  $(x', y')$  is a valid coordinate and no cut-out field
8      $result[x', y'] \leftarrow \text{getRandomFieldType}(x', y')$ 
9   end
10 return  $result$ 
```

Note that the method *getRandomField()* includes the guided mutation described in chapter 3.4, while *getRandomFieldType(x', y')* respects the limitations discussed in chapter 3.3, as well as the adjusted field type probabilities presented in chapter 3.4. Additionally - as long as at least two different field types are allowed at that position - a field type *different from the current type* is returned.

Algorithm 2: Crossover

Input: Parents $parent_1, parent_2$

```
1  $(g_x, g_y) \leftarrow$  central point with respect to only non-cut-out fields
2  $\beta \leftarrow$  random angle from  $[0, 2\pi)$ 
3 foreach field  $(i, j)$  do
4   | if  $(\sin \beta, \cos \beta)(i - g_x, j - g_y)^T \leq 0$  then
5   |   |  $result[i, j] \leftarrow parent_1[i, j]$ 
6   |   else
7   |   |  $result[i, j] \leftarrow parent_2[i, j]$ 
8   |   end
9 end
10 return result
```

Algorithm 3: Basic Hillclimber

Input: Initial Mask $mask$, Break Condition $limit$

```
1 evaluate(mask)
2 noChange  $\leftarrow$  0
3 repeat
4   | copy  $\leftarrow$  mutate(mask)
5   | evaluate(copy)
6   | if copy.rating < mask.rating then
7     |   | mask  $\leftarrow$  copy
8     |   | noChange  $\leftarrow$  0
9     | else
10    |   | noChange  $\leftarrow$  noChange + 1
11    |   end
12 until noChange  $\geq$  limit
13 return mask
```

Algorithm 4: Basic Genetic Algorithm

Input: Parameter n, α, c

```
1 for  $i \leftarrow 1$  to  $n$  do
2   |  $population[i] \leftarrow getRandomMask()$ 
3   |  $evaluate(population[i])$ 
4 end
5 sort  $population$  by descending rating
6 repeat
7   |  $newPop[1] \leftarrow population[1]$ 
8   | for  $i \leftarrow 2$  to  $n$  do
9     | if  $i \leq cn$  then
10    | |  $newPop[i] \leftarrow cross(select(population, \alpha), select(population, \alpha))$ 
11    | | else
12    | | |  $newPop[i] \leftarrow mutate(select(population, \alpha))$ 
13    | | end
14    | |  $evaluate(newPop[i])$ 
15    | end
16    | sort  $population$  by descending rating
17 until  $break\ condition\ is\ satisfied$ 
18 return  $population[1]$ 

19 Procedure:  $select(population, \alpha)$ 
20   |  $result \leftarrow population.selectRandom()$ 
21   | for  $i \leftarrow 2$  to  $\alpha$  do
22     | |  $m \leftarrow population.selectRandom()$ 
23     | | if  $m.rating > result.rating$  then
24     | | |  $result \leftarrow m$ 
25     | | end
26   | end
27   | return  $result$ 
28 end
```

Algorithm 5: Memetic Approach

Input: Parameter n, b_w, b_s, δ

```
1  /* Step 0: Initialization */
2  for  $i \leftarrow 1$  to  $n$  do
3  |  $population[i] \leftarrow hillclimbe(getRandomMask(), b_s)$ 
4  end
5  repeat
6  | /* Step 1: Crossover(s) */
7  | forall  $\{i, j\}$  in  $\binom{[n]}{2}$  do
8  | |  $m \leftarrow cross(population[i], population[j])$ 
9  | | for  $k \leftarrow 2$  to 50 do
10 | | |  $try \leftarrow cross(population[i], population[j])$ 
11 | | | if  $try.potentialRating > m.potentialRating$  then  $m \leftarrow try$ 
12 | | | end
13 | |  $newPop.add(m)$ 
14 | end
15 | /* Step 2: First Selection by potential Rating */
16 | remove all but best  $2n$  (with respect to potential Rating) masks from  $newPop$ 
17 | /* Step 3: First Hillclimber */
18 | for  $i \leftarrow 1$  to  $2n$  do  $newPop[i] \leftarrow hillclimbe(newPop[i], b_w)$ 
19 | /* Step 4: Delete masks that are too similar */
20 | sort  $newPop$  (by descending actual rating of masks)
21 |  $newPop[*].markedForDeletion \leftarrow false$ 
22 | for  $i \leftarrow 1$  to  $2n$  do
23 | | if  $newPop[i].markedForDeletion$  then continue
24 | | for  $j \leftarrow i + 1$  to  $2n$  do
25 | | | if  $newPop[i]$  and  $newPop[j]$  are too similar then
26 | | | |  $newPop[j].markedForDeletion \leftarrow true$ 
27 | | | end
28 | | end
29 | end
30 | /* Step 5: Second Selection by actual Rating */
31 |  $population \leftarrow$  all masks from  $newPop$  not marked for deletion
32 | if  $population.Count \geq n$  then keep only best  $n$  masks in  $population$ 
33 | else  $n \leftarrow population.Count$ 
34 | | /* Yes, the population size is actually reduced, if
35 | | there are not enough different masks left */
36 | /* Step 6: Second Hillclimber */
37 | for  $i \leftarrow 1$  to  $n$  do  $population[i] \leftarrow hillclimbe(population[i], b_s)$ 
38 until break Condition is satisfied
39 return best mask in population
```

B. Sample Masks

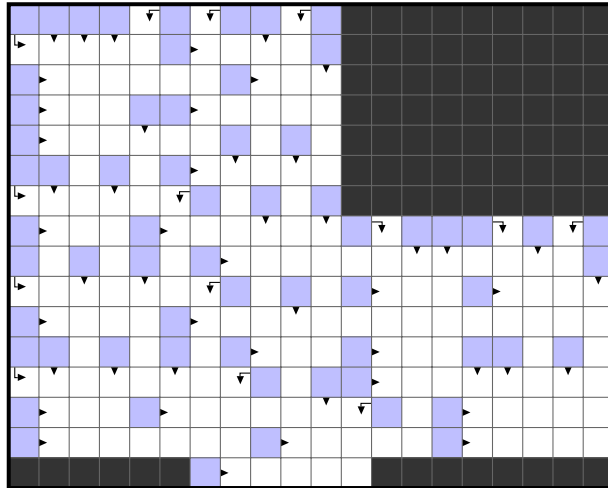


Figure B.1.: Manually created original mask (total penalty 10,458)

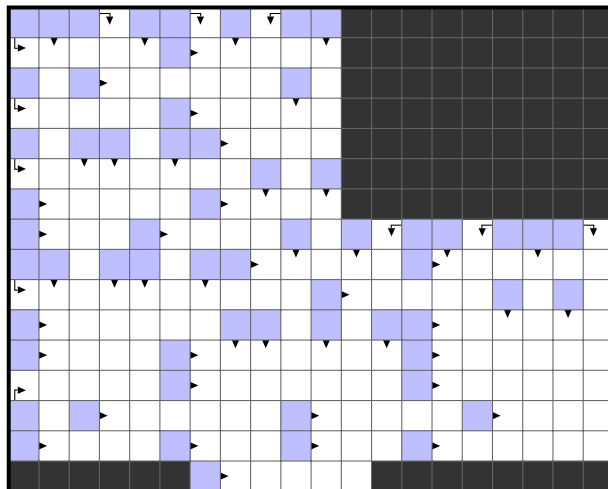


Figure B.2.: Automatically generated mask (total penalty 7,996)

Coverage:	Perfectly (5)	Once, enclosed (2)	Once, not enclosed (3)
Manually created	137	42	0
Generated mask	143	39	0

Word lengths:	2	3	4	5	6	7	8	9	10	11	12	13	14
Manually created:	0	17	14	13	10	6	2	0	0	0	1	0	1
Generated mask:	0	6	12	17	15	6	4	0	1	0	0	0	0

Table B.1.: Some basic statistics for the above masks

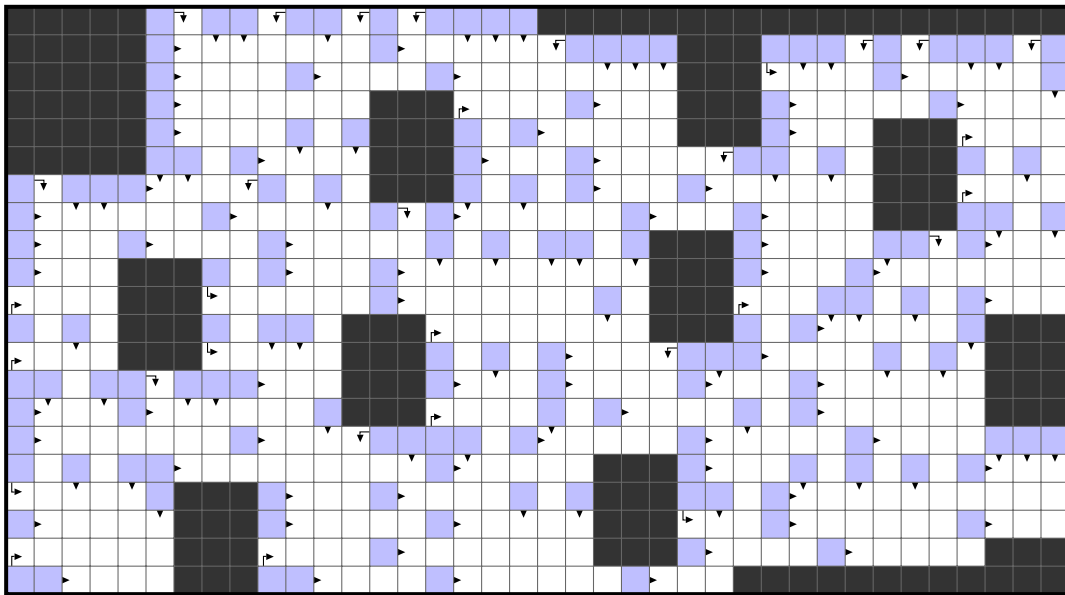


Figure B.3.: Manually created original mask (total penalty 37,322)

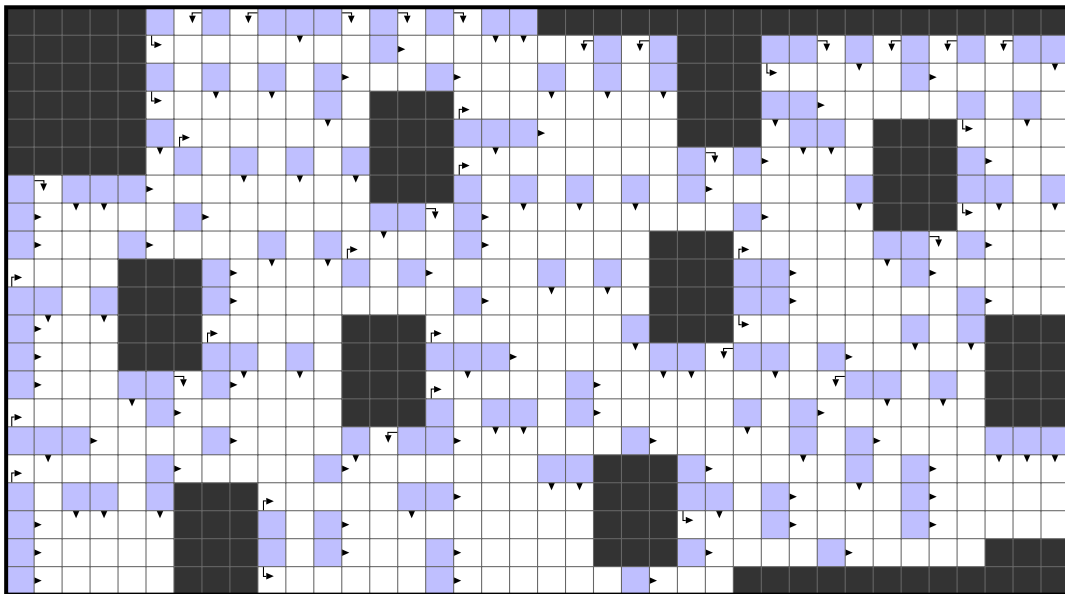


Figure B.4.: Automatically generated mask (total penalty 26,675)

Coverage:	Perfectly (5)	Once, enclosed (2)	Once, not enclosed (3)
Manually created:	354	102	0
Generated mask:	348	114	0

Word lengths:	2	3	4	5	6	7	8	9	10	11	12	13	14
Manually created:	0	51	42	26	21	15	10	3	1	1	0	0	0
Generated mask:	0	34	36	43	27	10	9	5	0	0	0	0	0

Table B.2.: Some basic statistics for the above masks

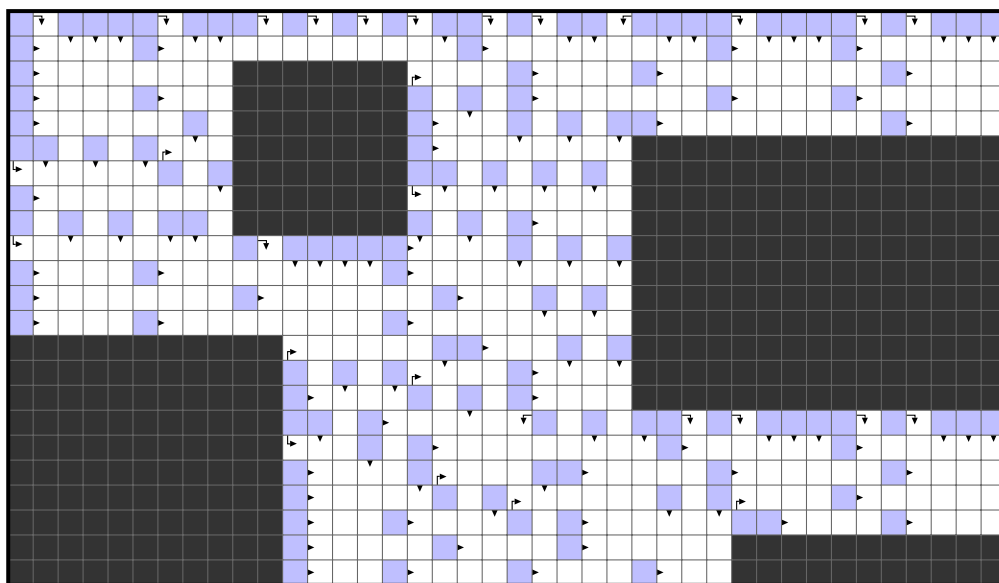


Figure B.5.: Manually created original mask (total penalty 29,824)

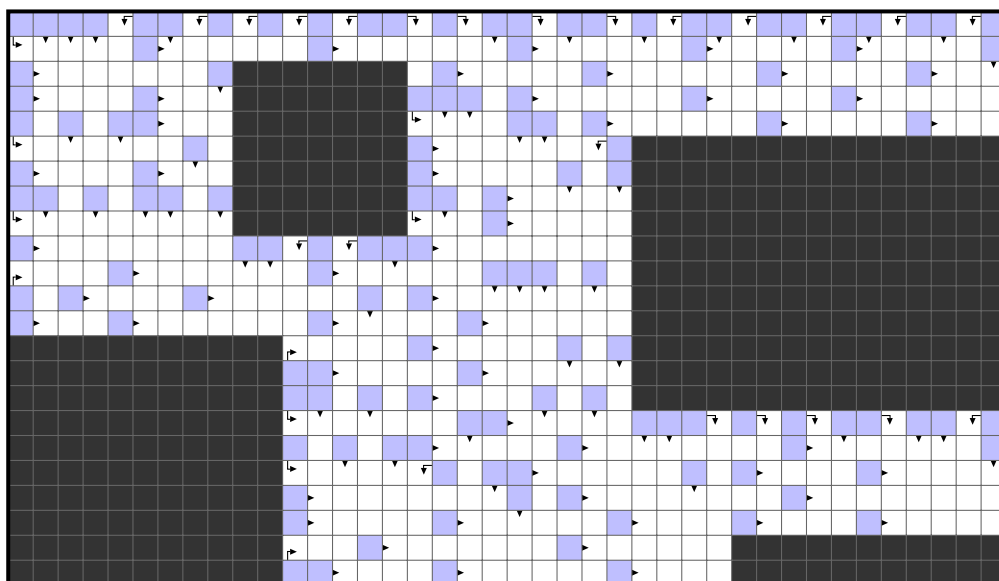


Figure B.6.: Automatically generated mask (total penalty 21,833)

Coverage:	Perfectly (5)	Once, enclosed (2)	Once, not enclosed (3)
Manually created:	330	93	0
Generated mask:	326	101	0

Word lengths:	2	3	4	5	6	7	8	9	10	11	12	13	14
Manually created:	3	20	64	16	22	8	5	11	0	0	2	0	0
Generated mask:	3	19	26	47	27	14	9	1	1	0	0	0	0

Table B.3.: Some basic statistics for the above masks

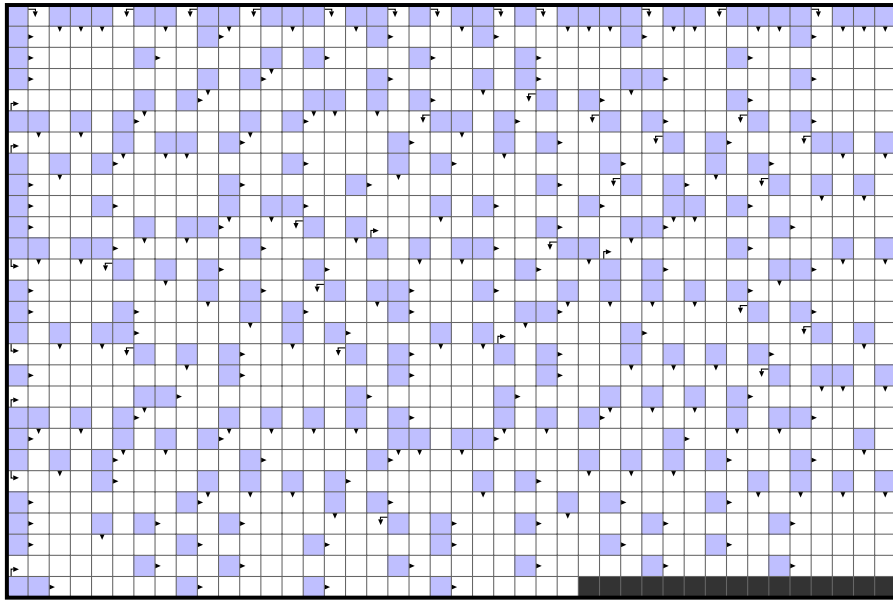


Figure B.7.: Manually created original mask (total penalty 47,819)

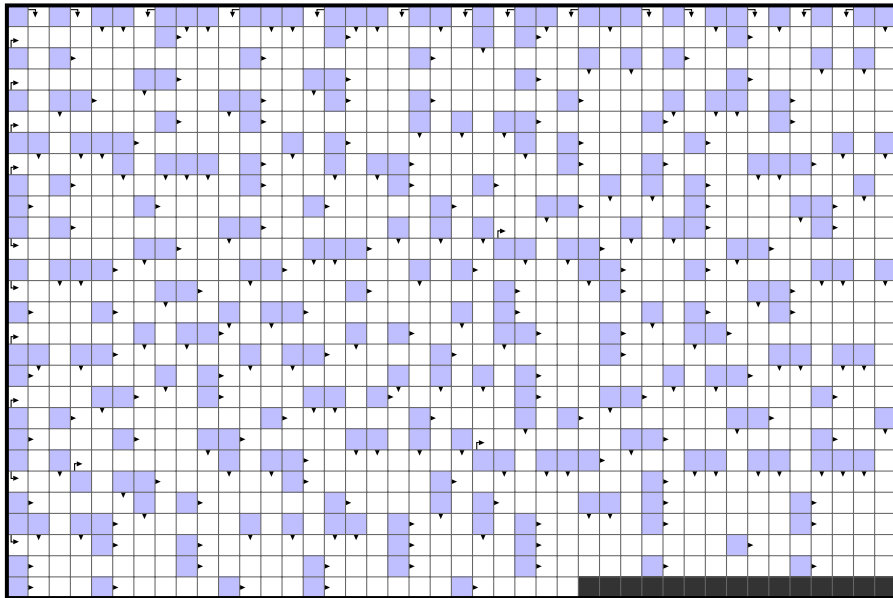


Figure B.8.: Automatically generated mask (total penalty 44,657)

Coverage:	Perfectly (5)	Once, enclosed (2)	Once, not enclosed (3)
Manually created:	672	203	0
Generated mask:	699	172	0

Word lengths:	2	3	4	5	6	7	8	9	10	11	12	13	14
Manually created:	0	26	74	70	45	31	27	10	3	0	0	0	0
Generated mask:	0	52	36	68	63	43	12	8	5	3	0	0	0

Table B.4.: Some basic statistics for the above masks

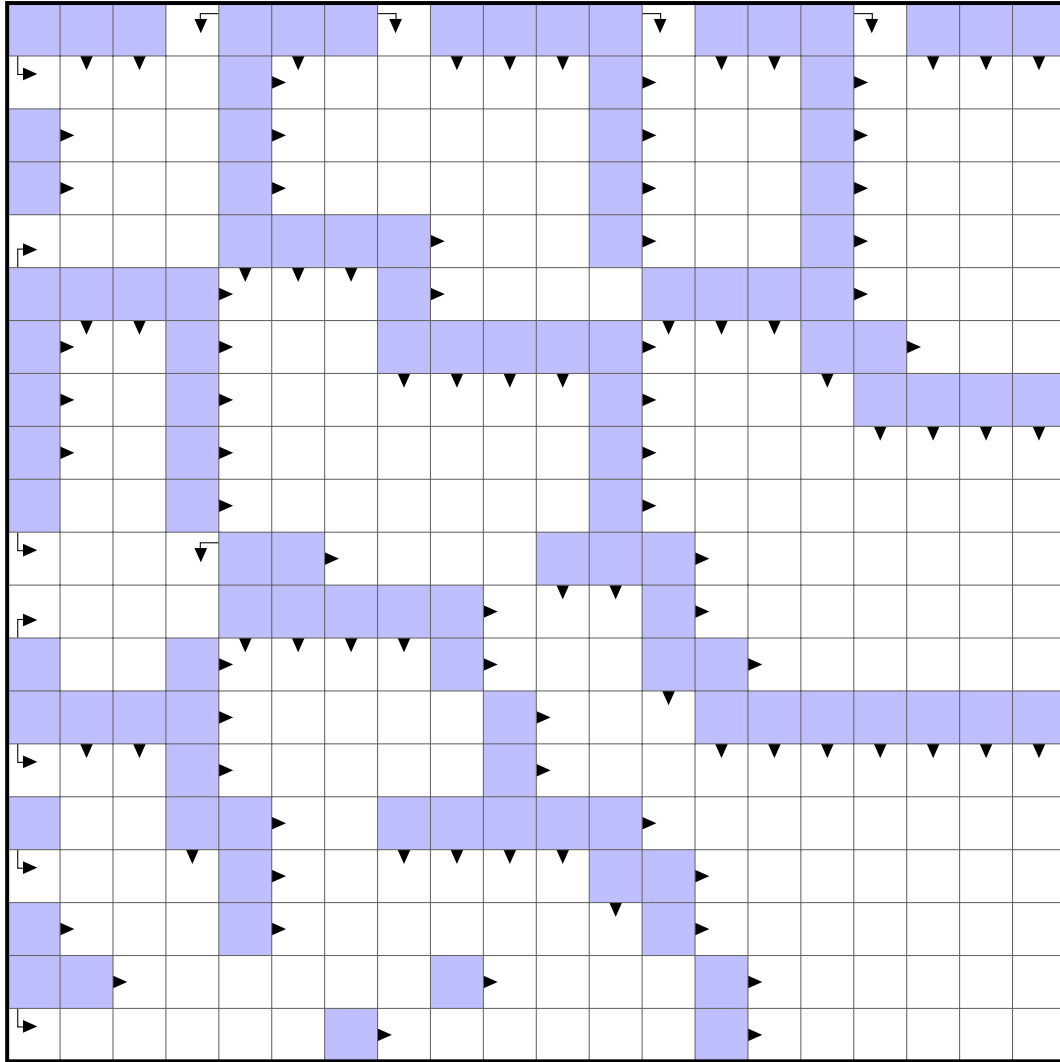


Figure B.9.: And this is what happens when large clusters are rewarded instead of penalized (i.e. the sign gets switched accidentally).

Bibliography

- [1] Michael Vose Alden, Michael D. Vose, Alden H. Wright, and Jonathan E. Rowe. Implicit parallelism. In *In GECCO (2003)*, pages 1505–1517, 2003.
- [2] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.
- [3] John H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, 1992. originally published in 1975.
- [4] R. Kruse I. Gerdes, F. Klawonn. *Evolutionäre Algorithmen*. Vieweg Verlag, 2004.
- [5] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, Caltech Concurrent Computation Program 158-79, 1989.
- [6] Oliver Ruepp. On the computational complexity of crossword puzzles - theoretical and practical considerations, 2005.
- [7] J. David Schaffer and Amy Morishima. An adaptive crossover distribution mechanism for genetic algorithms. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 36–40, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.
- [8] Wikipedia. Crossword - Wikipedia, the free encyclopedia, 2009.