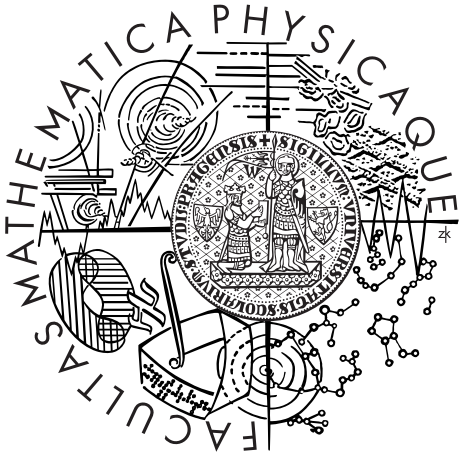
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Bc. Karel Tesař

Optimization Algorithms Inspired by Social Interactions

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	Mgr. Roman Neruda, CSc.
Study programme:	Computer Science
Specialization:	Theoretical Computer Science

Prague 2016

I thank to my supervisor Mgr. Roman Neruda, CSc. for the all valuable advice he gave me, for all content and grammatical comments he provided me with and for directing my work, thereby elevating it to a higher level.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... on ..... Author signature

Název práce: Optimalizační algoritmy inspirované sociálními interakcemi

Autor: Bc. Karel Tesař

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Roman Neruda, CSc.

Abstrakt: Existuje řada optimalizačních algoritmů, které se inspirují přírodou nebo chováním některých živočichů. Zároveň se v poslední době rozšířil výzkum sociálních sítí, který přináší výsledky o jejich strukturálních vlastnostech. Na základě inspirace z těchto oblastí navrhujeme optimalizační algoritmus sociální interakce. Algoritmus je založený na myšlence genetického algoritmu, jeho výpočet je řízen kombinatorickým grafem a operátory jsou inspirovány sociálními sítěmi a lidským chováním. Algoritmus přináší některé originální myšlenky a má svá vlastní specifika (jako nepoužití žádné selekce nebo možnost mít v populaci jedince různých typů) a je otevřen pro další rozšíření. V některých případech je efektivnější než ostatní algoritmy podobného typu, zejména ve finálních fázích běhu. V této práci představíme vlastnosti algorimu sociální interakce a demonstrujeme jeho výkon pomocí experimentů.

Klíčová slova: optimalizace, prohledávání, evoluční algoritmy, sociální sítě

Title: Optimization Algorithms Inspired By Social Interactions

Author: Bc. Karel Tesař

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: Mgr. Roman Neruda, CSc.

Abstract: A number of optimization search algorithms exists that are inspired by nature or by the behaviour of animal species. At the same time, there is an active research of social networks going on that brings us knowledge about their structural properties. We join an inspiration from both those areas and develop the social interaction algorithm. It is based on genetic algorithms, its computation is led by a combinatorial graph, and operators are inspired by social networks and by human behaviour. The algorithm brings some original ideas and has its own specifics (such as no usage of selections, or a possibility to have individuals of various types in the population) and is open to extensions. In some cases, it is more efficient than other algorithms of similar type, especially in the final phase of the run. In this thesis we introduce all features of the social interaction algorithm and demonstrate its performance by experiments.

Keywords: optimization, search, evolutionary algorithms, social networks

Contents

Chapter 1

Introduction

In computer science, optimization search is currently a widely researched area that has many applications in other disciplines like biology, medicine, chemistry, economics, or combinatorics. One of the related topics that has broken through in the past few decades is nature inspired optimization algorithms. There are plenty of optimization algorithms that take inspiration in concrete biological processes. We use such inspirations to develop algorithms which are then able to solve general optimization problems. There are also algorithms that take inspiration in real animals and mimic their natural behaviour in order to achieve success in solving an optimization problem. For example, there exists the ant-colony algorithm, the bee-colony algorithm, the firefly algorithm, the bat algorithm, the cuckoo search and more [1]. The majority of these algorithms has been found successful and have earned their place in the field of optimization search.

Our motivation is to go one step further and take inspiration in an animal species that currently leads in most of the world: the human beings. We believe that since humans have had a big evolutionary success, they have to be characterized by some significant structural behaviour that plays a key role in their success. We also believe that if we find that special kind of behaviour, we can transform it into a new optimization search algorithm that might be as successful as humans in the real world. In past decades a research of social networks arose which is supported by the fact that internet social networks spread all over the world and are being used rapidly [2]. In other words, we have access to more information and data about social networks than ever before. So, we have decided to focus on social networks, and to take them as our inspiration to develop a new optimization algorithm.

The main goal of this thesis is to develop a new optimization algorithm: the social interaction algorithm that is based on social network structures and studies. Since the social interaction algorithm is very complex, contains a high number of parameters, and is open to introducing many new features, we focus more on its general properties and parameter tuning for general cases rather than developing a very specialized version to solve a concrete problem. Another goal is to compare the performance and convergence properties to other existing models. Because the social interaction algorithm is designed to solve binary optimization problems, we compare the algorithm to other models that are designed to solve binary optimization problems as well, like genetic algorithms for example.

In the rest of this chapter, let us give a brief introduction about the structure

of the thesis and the content of single chapters.

In chapter 2 we give an introduction into optimization search and important factors of its analysis as a determination of a solution quality, complexity analysis, and exploration versus exploitation. Then, we present already existing optimization algorithms, and we pay a special attention to nature-inspired optimization algorithms. We present especially algorithms that are somehow related to the social interaction algorithm like genetic algorithms and their variants. At the end of the chapter we introduce few concrete binary optimization problems that we use in experiments in chapter 5.

In chapter 3 we give an introduction to social network structure and analysis. Because it is related to the graph theory, we introduce some graph theory definitions as well. We describe structural properties of social networks and pay attention to connections between individuals and to the strength of those connections. Then, we give a look at the information flow through the network and at social epidemics from a view of important kinds of individuals that play a key role in its arise [3]. Again, we focus especially on aspects of social networks that are related to the social interaction algorithm. We present other aspects as well but we omit them in more detailed description.

Chapter 4 is the main chapter of this thesis. We join the knowledge from both chapter 2 and chapter 3 to develop the social interaction algorithm. We take inspiration in social networks, especially in their structures, as well as in knowledge from other optimization algorithms. We describe all components of the social interaction algorithm together with all operators and parameters it contains. Then, we analyze the algorithm theoretically, discuss parameter choices, and pay a special attention to graph choices because the graph defines a computational structure of the algorithm. Then, we summarize the main properties which differentiate the social interaction algorithm from other models. One of these properties is a possibility of individuality, and we outline several options of how to use it. We summarize the relation of the social interaction algorithm to other models and we formally prove that we can mimic Hill climbing method by the social interaction algorithm. At the end we look at opportunities of parallelism and we prove that we can run the social interaction algorithm using $\mathcal{O}(\sqrt{\frac{m}{\Delta}})$ threads efficiently, where m is the number of edges in a graph and Δ is the maximum degree in a graph.

In chapter 5 we investigate the performance and convergence properties of the social interaction algorithm by running experiments. It would be tedious to deeply test all properties and features of the social interaction algorithm introduced in chapter 4 so we focus more on basic parameters tuning, graph choice and on general convergence properties. Some algorithm properties will still stay open for more detailed future research. We also compare the social interaction algorithm to other models. We compare especially the convergence properties and population variability. For all experiments we use the one-max problem and the SAT problem instances.

Chapter 2

Optimization Search Algorithms

One of the common goals of mathematics and computer science is to solve optimization problems. The optimization is a widely studied topic which is involved in many particular areas of current research, and has applications in many different fields like physics, economics, medicine, chemistry, artificial intelligence, et cetera. Therefore, we are motivated to develop methods that solve optimization problems. There exist a lot of methods with different approaches, and they are being widely researched recently. In this thesis we concentrate on one specific group of such methods which is called *the optimization search algorithms* and we will look at that area later in this chapter. First let us give a brief introduction into the optimization in general.

When we solve an optimization problem P, we have a set of feasible solutions F and an objective function $f(x)$ that maps solutions from F to real values. Our goal is either to maximize the value of an objective function, then we talk about a maximization problem, or to minimize the value of an objective function and then we talk about a minimization problem [4].

Since we can easily turn every minimization problem to a maximization one and vice versa by replacing the objective function $f(x)$ by $f^*(x) = -f(x)$ or by $f^{\Delta}(x) = 1/f(x)$. Both substitutions work in both directions.

Optimization problems can be divided into two major categories depending on whether the variables are continuous or discrete. Optimization problems with discrete variables are known as *combinatorial optimization problems* and those with continuous variables are known as *continuous optimization problems*. In a combinatorial optimization problem, we are looking for an object like an integer, a graph or a permutation from a finite set of feasible solutions F. In a continuous optimization problem we usually work with real variables and the set of feasible solutions F is infinite [4].

The standard form of an (continuous) optimization problem is the following [4]

Definition 1. *The* ***optimization problem*** *is*

$$\begin{gathered} \min_x f(x) \\ \text{subject to} \\ g_i(x) \leq 0, i = 1, \ldots, m \\ h_i(x) = 0, i = 1, \ldots, p \end{gathered}$$

where

- $f(x) : \mathbb{R}^n \to \mathbb{R}$ *is the* ***objective function*** *to be minimized over the variable* x*,*
- $g_i(x) \leq 0$ *are called* ***inequality constraints****, and*
- $h_i(x) = 0$ *are called* ***equality constraints****.*

Since we do not need the inequality and equality constraints and we will do maximization in most of the cases, we use a more simple definition which, in addition, fits to discrete problems as well.

Definition 2. *The* ***maximization optimization problem*** *is*

$$\max_{x \in F} f(x)$$
$$\text{where } f : F \to \mathbb{R}$$

where the domain F *of a target objective function* f *is called a* ***search space****.*

Let us note that the definition 2 covers the definition 1 since we can take into account all inequality constraints and equality constraints and cover them in a shape of the domain F.

Our general goal when solving an optimization problem is to search for the global maximum (or minimum) of an objective function f. If we are given a function f by the simple exact formula, then we can use the mathematical analysis to get all local optima, and simply choose the best [5]. But unfortunately, we often work with more complicated functions and in conditions that we are not given an exact description, and we are just able to compute the result value for a given feasible solution. Hence we are motivated to develop more sophisticated optimization methods. As we mentioned before, one category of such methods is *the optimization search algorithms.*

Optimization search algorithms usually try to evaluate an objective function in some points in a specific order. In sections 2.3, 2.4, 2.6, and 2.7 we describe concrete examples of such algorithms.

Since the essential topic of our research is a new algorithm for solving binary optimization problems, we describe all methods as they solve a binary problem even if they are originally designed to solve a more general one.

Definition 3. *A* ***binary optimization problem*** *is an optimization problem* P *in a form*

$$\max_{x \in \{0,1\}^d} f(x)$$

where $f : \{0,1\}^d \to \mathbb{R}$ *is an objective function of a problem* P*.*

2.1 Optimization Algorithms Analysis

When we develop optimization algorithms we need a way to compare and to analyze them. We do it by several criteria. The crucial one is the quality of the solution that our algorithm produces. We can define the quality as the following.

Definition 4. *Let us have an optimization problem P which has an optimum solution of value $f(x^*)$ and let us have a solution x of a value $f(x)$. Then we say that the* ***quality*** *of the solution x is $q(x) = \frac{f(x)}{f(x^*)}$ when we solve a maximization problem, and $q(x) = \frac{f(x^*)}{f(x)}$ when we solve a minimization problem.*

Obviously, the best possible quality is always 1. Let us note that the definition 4 works best for objective functions that are always positive or always negative because then the quality always fits into the interval $[0, 1]$. Since we always work with positive objective functions, this definition is sufficient for us.

Another criterion we pay attention to, during optimization algorithm analysis, is the number of evaluations of an objective function, that the algorithm has done during its computation. The reason is that an objective function evaluation is often the most demanding and the most time consuming part of the whole algorithm [1].

Definition 5. *Let us have a run R of an optimization algorithm A that produces a solution x. Then a value* `eval`(R, x) *is a number of evaluations of an objective function that an algorithm has done before it outputs the solution x.*

So, our goal is to develop such algorithm that outputs a solution of as good quality as possible, and do as few evaluations of an objective function as possible. But we still cannot omit other complexities of the algorithm like the time complexity and the memory complexity. For example, let us have an algorithm that evaluates an objective function just once, output the optimum solution but has an exponential time complexity. Such algorithm might be still useless since it might run for years. So we have to take care of standard algorithm complexities as well.

2.2 Exploration and Exploitation

Optimization algorithms can be also analyzed from the way they explore the search space. In essence, all optimization algorithms should have two key components: *exploration* and *exploitation* which are also referred to as *intensification* and *diversification* [6].

Exploitation uses any information obtained from the solved problem to help generate new and better solutions. This process as well as information is typically local. Therefore, it is considered to be a local search. For example, Hill climbing method from section 2.3 is a typical example of exploitation. It is always climbing up from a current point. Its advantage is the fast convergence but its disadvantage is that it often gets stuck in some local optimum because the final solution largely depends on a starting point [1].

On the other hand, exploration covers exploring new information on the global level. It can generate solutions with enough diversity and far from the current solutions. Therefore, the search is typically on a global scale. The advantage of exploration is that it is less likely to get stuck in a local mode, and the global optimality can be more accessible. However, its disadvantages are slow convergence and the waste of computational efforts because many new solutions can be far from the global optimality [1].

Our goal is to find a good balance of exploration and exploitation so that an algorithm can achieve good performance. Too much exploitation and too little exploration mean that the algorithm may converge more quickly but the probability of finding a true global optimum may be low. On the other hand, too little exploitation and too much exploration can cause the algorithm to wander around with very slow convergence. The optimal balance should mean the right amount of exploration and exploitation, which may lead to the optimal performance of an algorithm. Therefore, the good balance of these two components is crucially important [1].

However, finding such balance is still an open problem. In essence, the balance itself is a hyperoptimization problem because it is the optimization of an optimization algorithm. In addition, such balance may depend on many factors, such as the working mechanism of an algorithm, its setting of parameters, tuning and control of these parameters, and even the problem to be considered. Furthermore, such balance may not universally exist, and it may vary from problem to problem [1].

2.3 Hill Climbing with Random Restarts

The most basic and the most typical representative of all optimization search algorithms is the *Hill climbing method*. The idea comes from a real hill climbing. When we imagine a search space as a landscape we are walking on, then we always do a step that increase our altitude in order to reach the top. And that is exactly what the Hill climbing method does [7].

Mathematically said, let us have a d-dimensional real function f which we want to find a global maximum of. At the beginning we generate an arbitrary initial point x from a search space and then we iteratively do the following. We generate a new point $x' = x + \varepsilon$ where ε is a vector that contains a small random change in its every dimension. A vector ε represents a small step in the landscape. Then if $f(x') \geq f(x)$, we replace x for x', and we keep x otherwise. Finally, we continue with the next step.

The algorithm ends after the given number of iterations N or when the end criteria is reached, usually when the solution is close enough to the true global optimum. From the description above we may simply write an algorithm in pseudo code as in figure 2.1.

By the view of exploration and exploitation, Hill Climbing Method is a pure exploitation because we only use the last local point to obtain the next one and the final destination point strongly depends on the initial one. But we can extend it by some degree of exploration by executing the algorithm repeatedly several times with different initial points. Then, we take the best result from all executions and thereby increase the probability of reaching the true global optimum [1].

The described version of an algorithm is developed for d-dimensional real space. Now we modify it to get a version that works in d-dimensional binary space. Generating the initial solution can stay the same, again we just pick some initial point at random. But we have to change the way we pursue the small step. We cannot perform a random change in every dimension because then it is equivalent to generating a new random point. But we can, for example, flip one single bit at random instead, or rather to flip b random bits instead.

Algorithm 1: Hill Climbing Method

```
choose an initial point x from the search space at random;
t = 0;
while the end criteria is not reached do
    ε = random change;
    x' = x + ε;
    if f(x') ≥ f(x) then
        x = x';
    end
    t = t + 1;
end
```

Figure 2.1: The pseudo code of a hill climbing method

2.4 Simulated Annealing

One of the earliest and yet most popular optimization algorithms is *simulated annealing*, which is a trajectory-based, random search technique for global optimization [8]. It mimics the annealing process in materials processing when a metal cools and freezes into crystalline state. The annealing process involves the careful control of the temperature and its cooling rate. Simulated annealing has been successfully applied in many areas [1].

Again, as in section 2.3, we want to find a global maximum of a d-dimensional real function f. The idea of simulated annealing is very similar to Hill climbing method. At the beginning it picks a random point x and starts doing small steps around to point x' as well as the Hill climbing method does. But unlike the Hill climbing method, there is a chance to keep a new point x' even when it is worse than a point x. The probability of keeping a worse point is given by the current value of the temperature T and the size of a difference $\Delta f = f(x) - f(x')$.

In particular, the new point x' is preferred over x when either $f(x') \geq f(x)$ or $f(x') < f(x)$ and

$$p = \frac{\Delta f}{T} > r$$

where r is a random number, used as a threshold, which can be drawn from a uniform distribution $[0, 1]$ [9].

The temperature T we can take as a function of iteration number t which is called the *cooling schedule*. Two commonly used cooling schedules are linear and geometric. For a linear cooling schedule we have

$$T = T_0 - \beta t$$

where T_0 is the initial temperature and β is the cooling rate. It should be chosen in a way that $T \to 0$ when $t \to N$ (the maximum number of iterations). It gives $\beta = T/N$ [1].

On the other hand, geometric cooling schedule decrease the temperature by a cooling factor $0 < \alpha < 1$ so that T is replaced by αT which means $T(t) = T_0 \alpha^t$ [9].

Algorithm 2: Simulated Annealing Algorithm

```
1  choose an initial point x from the search space at random;
2  t = 0;
3  T = T_0;
4  while the end criteria is not reached do
5  |  ε = random change;
6  |  x' = x + ε;
7  |  accept = false;
8  |  if f(x') ≥ f(x) then
9  |  |  accept = true;
10 |  end
11 |  else
12 |  |  generate a random number r;
13 |  |  Δf = f(x) − f(x');
14 |  |  if exp(−Δf/T) > r then
15 |  |  |  accept = true;
16 |  |  end
17 |  end
18 |  if accept then
19 |  |  x = x';
20 |  end
21 |  t = t + 1;
22 |  T = αT;
23 end
```

Figure 2.2: The pseudo code of a simulated annealing algorithm.

Altogether we can write down the pseudo code of the algorithm as in figure 2.2. The algorithm ends after a given number of iterations or when a final temperature T_f is reached.

The advantage of simulated annealing against Hill climbing method is that it is able to overcome local optima. That is effective especially for functions with a ragged surface [1].

We may turn simulated annealing to solve binary problems instead of real problems in the same way like we did it for hill climbing method in section 2.3.

2.4.1 Parameters Control

Now we look more closely at how parameters influence the behaviour of an algorithm. Let us look at the temperature first. When T is too high ($T \to \infty$) then $p \to 1$ and the new point is "always" preferred over the old one and the algorithm becomes to be a simple random walk. On the other hand when T is too low ($T \to 0$) then $p \to 0$ and the algorithm is "equal" to Hill climbing method [1].

So, the initial temperature is crucially important and has to be carefully chosen due to the shape of function f. It is because the probability also depends on the value Δf.

As we have already mentioned, the parameter β is good to set $\beta = T_0/N$ in

linear cooling schedule. In the case of geometric cooling schedule, we must be careful that the temperature is not decreasing too quickly. The cooling process should be slow enough to allow the system to stabilize easily. Hence the good, commonly used, choice is $\alpha = 0.7 \sim 0.99$ [1].

The advantage of the geometric cooling schedule is that $T \to 0$ when $t \to \infty$, thus we do not need to specify the maximum number of iterations [1].

2.5 Nature-Inspired Optimization Algorithms

An optimization is a hard discipline, and we do not know the perfect solution yet. At the same time we can watch the behaviour of the nature. There are plenty of biological processes which are developed by billions of years of evolution, and which just works and fulfil its nature. So, there was a following question at the table: "Is it possible to simulate these biological processes and force them to help us to reach our goals in optimization problem solving?"

The answer seems to be "Yes." During the last thirty years there were developed many successful nature-inspired optimization algorithms, and it is still an area of an active research.

Nature-inspired optimization algorithms are often *population based* or *swarm intelligence based*. In both cases, there are individuals which represent single solutions of a problem. These individuals somehow interact with each other due to concrete algorithm's specifics. After many generation or many such interactions, there is, hopefully, developed an individual which is much better.

The most famous population based algorithm is the genetic algorithm described in section 2.6. It takes an inspiration in the evolution itself and it is good for solving optimization problems in binary search space. Swarm intelligence based algorithms as ant colony algorithm [10], bee colony algorithm [11] or firefly algorithm [1] take an inspiration in real animals and their interaction within colonies. Ant and bee algorithms are good to solve combinatorial problems and firefly algorithm is great to solve an optimization in a real space.

There exists much more nature-inspired optimization algorithms like bat algorithm, cuckoo search, flower pollination algorithm and many more. We work mostly with a genetic algorithm and its modifications. Some of these algorithms are described in a book Nature-Inspired Optimization Algorithms from Xin-She Yang [1].

2.6 Genetic Algorithm

The *genetic algorithm* is a population based algorithm and was invented by John Holland and his collaborators in the 1960s and 1970s [12]. It is a model of biological evolution based on Charles Darwin's theory of natural selection. Since that time a vast majority of well-known optimization problems have been tried by genetic algorithm. Many modern evolutionary algorithms are directly based on genetic algorithms or have some strong similarities [1].

We describe the genetic algorithm in its basic version which is developed to solve a binary optimization problem.

Genetic algorithm is population based search. A *population* is a set of individuals. An *individual* is represented as a binary string of *genes* that represents one point from a binary search space. Every individual has its *fitness value* which is computed by the fitness function F. The fitness function should be proportional to the target objective function f. A fitness of an individual determines how good is the solution represented in his or her genes. The better the individual's objective value $f(x)$ is, the higher the fitness value $F(x)$ should be. So, for the maximization problem the choice $F(x) = f(x)$ is good.

During a single iteration of an algorithm all individuals are gradually modified by three genetic operators: *selection*, *crossover* and *mutation*. These operators mimic the reproduction process in evolution, new individuals are created and the old ones die. Then the algorithm continues with the set of new individuals in the next iteration. Such one iteration of genetic algorithm is called *generation* [12].

Selection is an operator that selects individuals which will be involved in reproduction to the next generation. It is done according to fitness of individuals. We do not care whether we select one concrete individual more than once. When it happens, we just create more copies of him. There exist various types of selection, for example a *roulette wheel selection* or a *tournament selection*. In tournament selection we take two (or possibly more) individuals at random and we take the best to the next generation. We repeat that process until we select a given amount of individuals.

The roulette wheel selection picks a random individual. The probability that an individual is picked is directly proportional to his fitness value, concretely

$$p_i = \frac{F(x_i)}{\sum_j F(x_j)}$$

where x_j is a genetic code of an individual j. It is necessary to have positive fitness function in order to roulette wheel selection works correctly. Again we repeat the process as many times as many individuals we need.

The crossover of two parent individuals happens with a probability p_c. Then genes of these two individuals are combined and two new are created from them. There are also various kinds of crossovers. The most widely used are *one-point crossover* and *uniform crossover*. One point crossover choose a random point between bits and swap the second half of both individuals. For better imagination see figure 2.3. It follows the idea that close genes are more related, so they should not be split easily. On the other hand, a uniform crossover swap every bit independently with probability 0.5. The crossover operator is considered to be the main part of the genetic algorithm and usually has higher probability than mutation.

The mutation operator represents small random changes that can occur during evolution process, and happens with a probability p_m. We can have for example *bit flip mutation*, or *bit string mutation*. Bit flip mutation just flips one random bit in an individual. Bit string mutation flips every bit with a probability p_b. A value p_b is often set to $\frac{1}{d}$ where d is the length of an individual, so the expected number of flipped bits is one. Mutation usually has a smaller probability than crossover, and it should mainly helps the algorithm to get out from local optima.

Unlike a real evolution, we would like to keep the best individuals alive in order to the best fitness would not be decreasing. We realize it by the heuristic

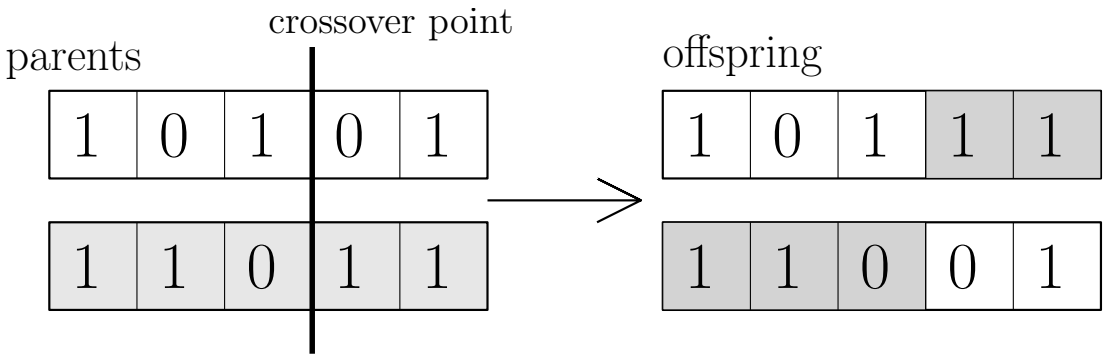


Figure 2.3: Example of one-point crossover from the genetic algorithm. Two parent individuals on the left side are combined and produce the offspring on the right side.

Algorithm 3: Genetic Algorithm

```
1  define the fitness F;
2  generate the initial population P and compute their fitness;
3  t = 0;
4  while end condition is not satisfied do
5  |  selection: choose individuals from P into new population Q;
6  |  for every pair of individuals in Q apply crossover with probability p_c;
7  |  for every individual in Q apply mutation with probability p_m;
8  |  compute fitness of individuals in Q;
9  |  add best E individuals from P into Q;
10 |  P = Q;
11 |  t = t + 1;
12 end
```

Figure 2.4: The pseudo code of genetic algorithm

that keeps the best E individuals for the next generation. This property is called an *elitism*. Elitism helps to a faster convergence of the algorithm because it can never happen that the best individual is either not selected or is modified badly. However, very strong elitism can cause premature convergence.

The algorithm starts with random generated individuals and ends after a given number of iterations or when the best possible individual is evolved. After joining everything together we can write down a pseudo code of an algorithm as in figure 2.4. It is recommended to run the algorithm several times with different initial data because different initial data can lead to different solutions of various qualities.

2.6.1 Role of Genetic Operators

As we already introduced, genetic algorithms have three main genetic operators: crossover, mutation and selection. Their roles can be very different [1]. Regarding to individuals represent solutions of a solved problem we talk about individuals as about solutions.

- *Crossover*. Swapping parts of the solution with another. The main role is to provide mixing of the solutions and convergence in a subspace. It uses already existing information contained in individuals.

- *Mutation*. The change of parts of one solution randomly, which increases the diversity of the population and provides a mechanism for escaping from a local optimum. It explores new informations.

- *Selection*. The use of the solutions with high fitness to pass on to next generations. An elitism can be also considered as a part of selection. Selection is the only operator which uses the fitness values of individuals.

Let us analyze the genetic algorithms by terms of exploration and exploitation. Exploitation component of an algorithm is composed by crossover and selection. They both work in local subspaces and together create new solutions from already existing ones. On the other hand, exploration component is composed by mutation and initialization of an algorithm. Both of them freely explore new informations that might be not known before [1].

All genetic operators together make the algorithm to be very complex. It is crucial to balance all the parameters to the algorithm works well and it is strongly dependent on concrete problem and fitness function realization [1].

It is worth pointing out that these genetic operators are fundamental. Other operators may take different forms, and hybrid operators can also work. However, these basic operators are important to understand the basic principles of genetic algorithms so we focus on them.

2.6.2 Choice of Parameters

An important issue is the choice of an appropriate fitness function that determines the selection criterion in a particular problem. For the minimization of a function, one simple way of constructing a fitness function is to use the form $F(x) = A - f(x)$, where A is a appropriately large constant.[1] Thus, the objective is to maximize the fitness function while minimize the objective function $f(x)$. Alternatively, for minimization problem, we can define a fitness function $F(x) = 1/f(x)$ but there may appear a problem when $f(x) \to 0$.[2] However, there are many different ways of defining a fitness function [1].

Choice of the fitness function should guarantee that that individuals with higher fitness are selected efficiently. Bad choice of fitness function may cause that incorrect or meaningless individuals are preferred.

Another important issue is the choice of various parameters. The crossover probability p_c is usually very high, typically in the range of $[0.7, 1.0]$. On the other hand, the mutation probability p_m is usually small, typically in the range $[0.001, 0.05]$. If p_c is too small then the crossover occurs sparsely, which is not efficient for evolution. If the mutation probability is too high, the solution could still "jump around," even if the optimal solution is approaching [1].

[1] We set $A = 0$ if the positive fitness is not required

[2] Especially when we use roulette wheel selection. In case of tournament selection it is not a problem while we do not reach zero.

A proper criteria for selection the best solution is also important. How to select individuals to pass to the next generation is a question that is still not fully answered. Selection is often supplemented with elitism. The basic elitism is to select the fittest individual (in each generation) which is passed to the new generation unmodified by genetic operators. This ensures that the best solution is achieved more quickly [1].

Other issue is to choose a proper population size. If the population is too small then there is not enough of genetic information and there is not much evolution going on. If the population size $n \to 1$, the algorithm becomes to be just simple random walk. In the real world, for species with a small population, ecological theory suggests that there is a real danger of extinction for such species [1]. In addition, in genetic algorithm with small population, when a significantly fitter individual appears, there is a danger that its offspring overwhelm the whole population soon. On the other hand, too large populations needs more evaluations of fitness function which costs us additional computation time. Studies and empirical observations suggest that the population size $n = 40$ to 200 works for most problems [1].

2.7 Genetic Algorithm Variants

Because of the success of genetic algorithms, its new variants are still being developed and many other algorithms are based on the idea of genetic algorithms. In this section we focus on modifications that investigate a structural behaviour of individuals within the population. That means that individuals are not allowed to interact with everyone all the time, as it is done in a standard genetic algorithm, but their interactions are directed or bounded by the, so called, *population model*. We introduce two population models in this section: *the island model* and *the neighbourhood model*.

2.7.1 Island model

The island model is a genetic algorithm modification where individuals are divided to a separate "islands" lying in the circle and on each of them there is an evolution process going on like in the genetic algorithm [13]. Once in a while (after given number of iterations) the best individual from every island is taken and sent to the right neighbouring island. Such moving of individuals is called the *migration*.

Every island evolves its unique kind of solutions and carry different genetic information. Migration, in essence, takes the best individuals from different runs of the genetic algorithm and enrich them by genes from other population. There were studies which showed that such migrations help to faster convergence, and that it leads to performance improvement which will not be fully realized by models that has all subpopulations (islands) independent [14].

An issue of the island model is to decide how often to do the migration (it is called the *migration rate*). It was shown that an island model with relatively small migration rate causes acceleration of the rate of development of the best individual [15].

Island or migration models described above are also referred as *panmictic* genetic algorithms.

2.7.2 Neighbourhood model

The neighbourhood model supposes that each individual has its own geographical location. Then, every individual can move just in his bounded region. The individual may interact only with those of other individuals that are living in immediate locality which we call the *neighbourhood*. The neighbourhood is determined by the topological structure of a manifold that individuals live at. We may see the neighbourhood as the set of potential partners of a individual. We call them *friends* [16].

Individuals are placed regularly on a manifold so the graph of friendship has a "grid" structure. We can notice that since it is a connected graph, genes informations can propagate through the whole graph when we let individuals to interact long enough. In some literature it is referred as a *diffusion process* [16].

Let us see how the neighbourhood model with a graph $G = (V, E)$ works exactly.[3] We have an individual sitting in every vertex of G. In a single iteration every individual does the same: selects one of its friends, uses its copy for the reproduction (crossover and mutation), then the offspring is created, the *survival strategy* is applied, and eventually an individual is replaced.

Selection, crossover and mutation can be the same as in the standard genetic algorithm (but selection uses just individuals from the neighbourhood). The survival strategy specifies which offspring (if any) is passed to the next generation. Five survival strategies differing in the strength of selective pressure have been defined in [16]:

- **A:** *accept all* – each offspring is accepted
- **B:** *1% worse* – accept only offsprings which are fitter than the local weakest + 1% (small degenerations are possible)
- **C:** *local least* – accept only offsprings which are fitter than the local weakest
- **D:** *average* – accept only offsprings which are fitter than the local average
- **E:** *parent* – accept only offsprings which are fitter than the local parent

The survival rule may be modified to be an *elitist strategy*:

> If the local parent has the best quality within its deme, i.e. is the local-best, than its offspring is accepted only if it is better than the parent [16].

The elitist strategy preserves all local-best individuals and consequently also the globalbest individual. By definition survival strategy E is always an elitist strategy [16]. In experiments in chapter 5 we always use the survival strategy C

[3]Let us note that originally in [16] the neighbourhood model was invented, and tested only for a toroid grid and a ladder. However, we can use any kind of a graph. Toroid grid and ladder graphs are described in more details in section 4.3.

Algorithm 4: Neighbourhood Genetic Algorithm

```
initialization of an algorithm and individuals;
t = 0;
while stop criteria is not satisfied do
    for every individual independently or in parallel do
        select a friend from neighbourhood by selection;
        reproduce (crossover and mutation) and create offspring;
        apply survival strategy and eventually replace the individual;
    end
    t = t + 1;
end
```

Figure 2.5: Pseudo code of the neighbourhood genetic algorithm.

with a global elitism (the globally best individual always survives and the local best not necessarily).

Let us show the pseudo code of an neighbourhood genetic algorithm in figure 2.5.

2.7.3 Analysis and Comparison of Models

Both island model and neighbourhood model can be treated by parallelism. In island model every island can run independently in parallel until the next migration time is achieved. The neighbourhood model can be even more parallel because its every individual can run independently from others.[4] They can send information about themselves to their friends by "mail boxes" and that is where their friends pick them up at the time they need. So, the neighbourhood model can be treated in a massively parallel way [16].

Let us refer to the comparison of the performance of standard genetic algorithms, panmictic genetic algorithms and genetic algorithms with diffusion process. In [17], there was concluded that genetic algorithms with diffusion process are more suitable for complicated functions than panmictic genetic algorithms. A study [18] compared all mentioned kinds of genetic algorithms over several benchmark problems. It was shown that panmictic genetic algorithm performs better than standard genetic algorithms, and genetic algorithms with diffusion process performs the best of all [16].

2.8 Concrete Optimization Problems

In chapter 5 we introduce several experiments and compare various algorithms by solving concrete optimization problems. The aim of this section is to introduce to all of these problems.

[4] In fact, it does not matter whether every individual is in the same generation or not.

2.8.1 One-Max Problem

One of the simplest binary optimization problems is to evolve a vector of ones [19]. The problem is mainly used to ensure that our algorithm is working correctly, and is able to end up with a vector of ones from a random initial data in a reasonable amount of time. The problem also can be used to obtain an initial setting of various parameters when developing a new algorithm.

2.8.2 SAT Problem

In computer science, *boolean satisfiability problem* is the problem determining if there exists an interpretation (evaluations of variables by `true` and `false`) that satisfies a given Boolean formula. *SAT problem* is a special kind of boolean satisfiability problem where the formula is in *conjunctive normal form* (CNF).

Definition 6. *The Boolean formula φ is in a* ***conjunctive normal form*** *when it is in a form*

$$\bigwedge_i \bigvee_j x_{i,j}$$

i.e. it is a conjunction of disjunctions where disjunctions we call ***clauses****.*

For every Boolean formula φ it is possible to find an equal formula ψ in CNF form with the same set of variables. Here equal means that they are equally satisfiable. Thus, for every evaluation of variables by `true` and `false` the formula φ is satisfied if and only if the formula ψ is satisfied.

The SAT problem was the first known NP-complete problem, (it was proven by Stephen Cook in 1971 [20]). Because of the fact that the SAT problem is NP-complete, it is possible to formulate any instance of any NP problem as an instance of the SAT problem. The SAT problem is widely studied and many heuristic algorithms are developed to solve it.

We look at the SAT problem as at a binary optimization problem. Our goal is to satisfy as many clauses as possible. Let us note that a clause is satisfied if and only if any of its literals acquire `true` value.

Chapter 3

Social Networks and Epidemics

Humanity forms one huge social network with its very specific structure. People are connected to others by strong and weak connections, and are living as a part of several different social groups. They are interacting with each other, influencing themselves and in that way getting a life experience. In such networks we are interested in: how the structure itself looks like, why it does look like that, how it is formed and divided into small or big groups, what is the information flow through the network, who are the important people, what roles a single person plays, et cetera. We can apply such analysis in social media, marketing, or during a political campaign. Other topics are related to epidemics (social epidemics and diseases as well), and how they begin, spread, or what is the flow of informations.

During past five years social networks become extremely popular, with over one billion of users on Facebook alone and billions more accounts accross thousands of social networking sites online [2]. Since social networks are widely used through the internet, they have become very popular and useful topic to research.

In this chapter, we mainly focus on the structure of networks and on social epidemics that are spread mostly by individuals. Then, in chapter 4 we apply the theory and observations to design an optimization algorithm.

We follow the notation and nomenclature from [2] in case of a social network structure and we follow nomenclature from [3] in case of social epidemics. But sometimes we slightly change it in order to obtain better understandability and readability for computer scientists.

3.1 Basic Terms and Definitions

For us, a social network is a standard combinatorial *graph* composed by *vertices* (*nodes*) that are connected by *edges*[1]. Vertices represent persons, and two persons are connected by the edge when they know each other (or i.e. are friends on Facebook). There might be an evaluation of an edge given by how well persons know each other but we usually do not have such information, so we work just with an unweighted graph structure containing no additional information about vertices or edges. We consider only undirected graphs. There might exist also directed edges in social relations but for simplicity we omit them as well as we omit loops and multiedges.

[1] In social network analysis edges are often referred as *ties*.

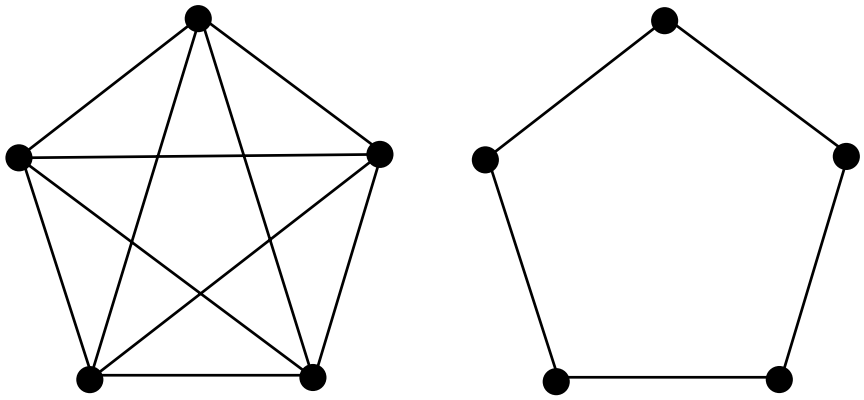


Figure 3.1: A clique K_5 (left) and a circle C_5 (right).

We say that a *distance* between vertices $u, v \in G$ is the number of edges on the shortest path between u and v. If there is no path between those two vertices then we consider the distance to be infinity.

We will often use the following special types of graphs: *clique* K_n and *circle* C_n. A *clique* is a graph with all possible edges and circle is a graph where vertices are connected just to their neighbours "in a circle", see figure 3.1. When we talk about a *subgraph* of a graph, we always mean an induced subgraph.

To describe a connectivity of a single vertex v we use a *vertex degree* $\deg(v)$ in its standard meaning (the number of edges incident to the vertex v. A vertex degree is a good way to describe how much a single vertex is connected to others. If we want to describe how the whole graph (or subgraph) is connected, we do it by *density* [2].

Definition 7. *Let G be a graph with n vertices and m edges then we say that a* ***density*** *D of G is*

$$D = \frac{2m}{n(n-1)}$$

where $\frac{n(n-1)}{2}$ is the number of possible edges.

When a density of a graph G is "high" ($D \to 1$) then we say that a graph is *dense* and when a density is "low" ($D \to 0$) then we say that a graph is *sparse*.

Once we have defined density, we can talk about clusters. For us, a *cluster* is a subgraph C of a graph G that is dense. It is not necessary for a cluster to be a clique. The exact definition of a cluster is not absolutely clear as it is not clear what clusters a given vertex belongs to. There can be a lot of clusters present in a graph of various densities and sizes. A single vertex can belong to more different clusters. In general, we want clusters to be as big as possible while they are still dense enough. Dividing a graph to clusters is crucial while analyzing the social network and ways how to do it are still being researched [21].

The last term we define is an *egocentric network*. Let us define it as follows

Definition 8. *Let G be a graph and v one of its vertices. Then an* ***egocentric network*** *of v in G is a subgraph G_v of G that contains a vertex v and all its neighbours.*

An egocentric network is used to analyze a role of an individual within the network and his or her belonging to different social groups [2].

3.2 Social Network Structure

We can characterize the most of social networks as sparse highly clustered graphs with small average distance between vertices.

The whole network usually contains huge amount of people while single person has only limited number of acquaintances. That is why the graph is sparse. On the other hand, it contains many dense subgraphs (clusters) which match various social groups like school classes, colleges at work, family, free time activity groups, et cetera.

A common phenomenon in a social media is the so called *six degree of separation*. It is a hypothesis proposing that any two people in the world are separated by short paths, on average about six steps. First it was demonstrated by Milgram in 1967 [22]. He made an following experiment: he has given a package to a random person in USA with instructions to deliver that package to another random person in USA. In one step, it was possible either to deliver the package to the target person directly or to pass it to someone who is more likely able to deliver the package. The average number of steps of packages that reached the destination was very close to six.

Within a Facebook network a phenomenon is even stronger and there is only three and half degrees of separation, as Facebook published at the beginning of 2016 [23].

3.3 Tie Strength

As authors of [2] stated: "Social relationships are complicated. The type of relationship people have will draw on many things like their history and similarity, each persons personal background and preferences, environmental factors, and more. Relationships are also multifaceted, and many relationship types can be used in social network analysis. One of the most useful is the idea of tie strength."

Tie strength is a measure of the strength of relationship between people [2]. The concept was introduced by Mark Granovetter 1973 [24], he stated: "the strength of a tie is a ... combination of the amount of time, the emotional intensity, the intimacy (mutual confiding), and the reciprocal services which characterize the tie."

Later researchers investigated which other factors might also play a role in tie strength. They concluded that except of *time*, *emotional intensity*, *intimacy* and *reciprocal services* there play a role also *structural features*, *social distance* and *emotional support*.

3.3.1 Structural Features

Since it is very hard to measure most of factors above, and they are dependent on data that are often not publicly available, and since we focus on the network structure, we only look at structural features closer and we try to estimate the tie

strength just by this one factor. But as we will see, all the factors are somehow binded and the structural features are often consequences of other factors.

We base the structural measurement on a simple idea. When two individuals have many mutual friends then they likely know each other better. The more mutual friends they have, the bigger is probability that they see each other on regular basis, the more mutual social groups they are part of and hence the socially closer they are.

So now we can estimate the tie strength by the number of mutual friends. We formulate it in terms of graph theory.

Definition 9 (Tie strength)**.** *For a given graph G and an edge $\{u, v\} = e \in G$ we say that a **tie strength** of e (a weight of e) is given by the expression*

$$|N(u) \cap N(v)|$$

where $N(w)$ is a set of all neighbours of w and vertex w itself.

3.3.2 Strong and Weak Ties

When a tie strength is high, then we say that it is a *strong tie*, and when a tie strength is low then we say that it is a *weak tie*. Strong ties are rare and indicate usually family members or very close friends. On the other hand, weak ties are much more common and include acquaintances and more casual friendships. Of course, there is a more accurate spectrum of tie strength but for simpler analysis we assume just these two major kinds [24, 2].

The tie strength is a very important factor to consider when analyzing social networks and an information flow. Strong ties are usually more trusted and weak ties usually has bigger chance to bring us new information since people on weak ties often come from different social groups. For example in 1973 Granovetter in his study "The Strength of Weak Ties" [24] researched how people are getting their jobs and concluded that weak ties play a crucial role and significantly many people get a job just by using a weak tie than doing anything else.

Similarly in replication of Milgram's "six degrees" experiment, researchers gave booklets to participants and instructed them to pass the booklets on until they reached an unknown target person. At each step the participants recorded to whom they gave the booklet and how they knew that person. Results showed that chains where the booklet successfully reached the target made much heavier use of weak ties [2].

Two experiments above shows the importance of weak ties. But it does not mean that strong ties are unimportant. Strong ties are much more trustworthy and reliable than weak ties.

3.4 Social Epidemics

Social epidemics spread in a network similarly like disease epidemic does but they have their own specifics in addition. A social epidemic can be a fashion trend, an emergence of best seller, smoking of teenagers, local criminality, et cetera. It spreads in a social network without any central control or a master plan.

One possible point of view at social epidemics is by strong and weak ties which we introduced in section 3.3.2. Now we look at another theory that comes from a book Tipping Point by Malcolm Gladwell [3]. He proposes that the following three factors are important for a social epidemic rise: *the law of the few*, *the stickiness factor* and *the power of context*.

Let us look closer at the law of the few. Gladwell in [3] stated: "Contagious expansion of ideas or systemic changes does not rely upon thousands or millions of people all rising up of one accord to create the change. Instead, the rapid growth is usually started by a handful of people who exhibit some kind of exceptional behaviour..." There are three important kinds of people for the social epidemic rise: connectors, mavens and salesmans.

Connectors are people who "know everyone." They know exceptional amount of people from many different social groups. They are usually also very popular and good in communication. Connector is likely one of the first who come in contact with a new information and when the information engages his interest, he or she distributes it to huge number of other individuals [3]. In other words, we can characterize a connector as an individual with a big egocentric network that connects many different clusters (or social groups).

Mavens are information specialists. They know everything there is to know about a certain topic (like sales in hypermarkets, new technologies, et cetera) and they love to share what they know with others. They are usually not connected that much as connectors but they are eager to share what they know. Mavens are important individuals. They are acquiring new informations that the others do not know. In a network they are likely the first to know a potential system change. If they are in touch with connectors, then the change can get communicated very rapidly [3].

Salesmans are people who can easily persuade others and get people to make decision and take action that they ordinary would not take. They can use their emotions as contagious influences on other people. Their ability to persuade makes them strong carriers of infectious ideas, concepts, trends and changes.

3.5 Chapter Conclusion

The purpose of this chapter is to provide inspirations from the social network analysis that we use in chapter 4 for developing the social interaction algorithm, and to understand the social network specifics because then we can get better understanding of some features of the algorithm and vice versa. We use structural properties when designing a graphs for the algorithm, we apply a tie-strength in experiment 5.6. In section 4.5.3 we transform special kinds of individuals described in section 3.4 into ideologically analogical individuals in the algorithm.

Chapter 4

Social Interaction Algorithms

In this chapter we take humans as the source of inspiration and we apply observations about their behaviour when solving optimization problems. We want to examine whether it can be useful to be inspired by animals in such high evolutionary stage.

We use the theory of optimization search introduced in chapter 2, knowledge about social networks referred in chapter 3 and combine them both in order to design a new algorithm. Then in chapter 5 we support the theory by results of experiments and we compare the performance to other algorithms.

The classification of *social interaction algorithm* is somewhere between population based and swarm intelligence based algorithms. Ideologically, the closest algorithm (known by us) is the neighbourhood model of genetic algorithm described in section 2.7.2 because they both lead the computation by some graph which defines what pair of individuals can interact to each other. Social interaction algorithm is based on the basic idea of the genetic algorithm itself. That is why we consider it to be population based. But we consider it to be swarm intelligence based as well because there is a group of humans in the environment that interact to each other according to given specific rules and the whole group tries to solve an optimization problem. So, the situation is similar like with a group of bees, or a group of ants, et cetera.

We design the algorithm especially to solve binary optimization problems. We test the performance on one-max problem and SAT problem instances. Our goal is to be able to solve tough SAT problem instances successfully, so we designed few modifications that fits SAT problem the best when it is not necessarily possible to apply them to any binary optimization problem with the same success. However the basic version of an algorithm and the most of its modifications should be applicable to any kind of binary optimization problem.

4.1 Algorithm description

Social interaction algorithm is a genetic-like algorithm. As in genetic algorithm we are given a binary function $f : \{0,1\}^d \to \mathbb{R}$ to optimize (maximize or minimize) which we turn to a positive fitness function F to maximize. In addition, we are given a graph $G = (V, E)$ with n vertices and m edges that defines a structure of population that we can understand as a social network. Every vertex represents an individual, and there is an edge between two individuals if they

are friends (they trust each other). In the algorithm we allow interactions only between individuals that are in friendship (are connected by an edge).

Every individual i represents a binary vector x_i and we imagine that values of a vector mimic individual's opinions on some fixed set of yes/no questions. As in the real world, if two individuals meet then, since they are friends, they have a discussion about several topics and there is a chance that one of them change a part of his or her opinions according to the other one and vice versa. In addition, there is also a chance that an individual change his or her opinion independently just by himself (i.e. from newspapers, books, television, et cetera).

More formally, a social interaction algorithm runs in time steps. In every single time step two operators can happen: *meeting operator* and *mutation operator*. Meeting operator happens during time step with higher probability p_c on a random edge $e \in E(G)$ and a mutation operator happens with lower probability p_m on a random vertex $v \in V(G)$. As a mutation operator we can use any kind of mutation we know from genetic algorithms. We will mostly use the single bit flip mutation. Meeting operator defines a way of discussion and changing opinions of individuals. We describe it more precisely in a section 4.1.2. Let us point out that there is no explicit selection in social interaction algorithm, and pairs of individuals to interact are fully determined just by the structure of graph G. The absence of selection is the underlying difference from genetic-like algorithms, since the selection operator is their very important part.

Algorithm runs for a given number of steps or until the best individual is achieved. We measure a running time by the number of fitness function evaluations which is the most significant part of an algorithm from the perspective of time consumption. We can write down the pseudo code of social interaction algorithm as in figure 4.1.

4.1.1 Mutation operator

Mutation represents small random changes in individuals. The goal is the same as in genetic algorithms: to discover new informations that have not occurred in the population yet. It fully belongs to exploration component.

Mutation happens with a lower probability p_m. Good choice usually is $p_m \in [0.001, 0.1]$. We can use any mutation as in genetic algorithm since it is dependent only on one single individual and nothing more. However mutation is not a crucial part of our research so we mostly use simple single bit flip mutation with probability $p_m = 0.1$. Single bit flip mutation flips exactly one random bit of an individual.

4.1.2 Meeting operator

The essence of a meeting operator is to share knowledge between individuals and mix them together in order to get better combination of opinions. It forms the exploitation component of an algorithm. The description reminds a classical crossover from the genetic algorithm but it cannot be as simple as that. There is no explicit selection in the social interaction algorithm and hence no warranty of "good" individuals meeting. So individuals have to take care about themselves and have changes of their opinions under control at least partially. Let us describe

Algorithm 5: Social Interaction Algorithm

input : A graph G of n vertices, fitness function F, max iteration IT
output: Genes of the *best* individual seen

```
1  initialize individuals and count their fitness;
2  best = an individual with the highest fitness from individuals;
3  reset evalCounter;
4  while evalCounter < IT and best.fitness < maxFitness do
5      if succeed with probability p_c then
6          perform meeting operator on a random edge {u, v} ∈ E(G);
7          recompute fitness of individuals u and v;
8          update the best;
9      end
10     if succeed with probability p_m then
11         perform mutation operator on a random individual v ∈ V(G);
12         recount fitness of an individual v;
13         update the best;
14     end
15 end
16 return best
```

Figure 4.1: The pseudo code of social interaction algorithm.

it precisely.

Meeting operator happens with a higher probability p_c and uses individuals a and b from endpoints of a random edge $e \in E(G)$. Individuals a and b have a discussion. They discuss every topic with a probability p_f which we call a *meeting factor*. In a single topic an individual a change his or her opinion according to individual b with probability

$$\frac{F(b)}{F(a) + F(b)}$$

and an individual b change his or her opinion according to individual a otherwise. By value $F(x)$ we mean a fitness value of individual x before the discussion started.

When the discussion is over, both individuals recount their fitness. For $x \in \{a, b\}$ if a fitness value of individual x does not decrease then he or she keeps all new opinions. If a fitness of individual x decreases then he or she goes back to his or her original opinion with probability p_s independently for every discussed topic. Probability p_s is given by the *self confidence* of an individual x. See the pseudo code of meeting operator in figure for better imagination 4.2.

Meeting operator is the main drive of social interaction algorithm so we usually choose probability $p_c \in [0.9, 1.0]$. Let us notice that for the performance, the important factor is the ratio of probabilities p_m and p_c rather than their concrete values. The best is to have p_m or p_c equal one to prevent "empty" iterations. But we usually use $p_c = 0.9$ and $p_m = 0.1$ to get more trustworthy comparison to other algorithms running with the same parameters.

Algorithm 6: Meeting operator

input: Individuals a and b

```
1  copy_a = a.copy();
2  copy_b = b.copy();
3  changes = empty list;
4  for i ∈ {1, ..., d} do
5      if succeed with probability p_f then
6          if succeed with probability F(b)/(F(a)+F(b)) then
7              a.genes[i] = b.genes[i];
8          end
9          else
10             b.genes[i] = a.genes[i];
11         end
12         changes.add(i);
13     end
14 end
15 for x ∈ {a, b} do
16     if F(x) < F(copy_x) then
17         for c ∈ changes do
18             x.genes[c] = copy_x.genes[c] with probability p_s;
19         end
20     end
21 end
```

Figure 4.2: The pseudo code of meeting operator.

4.1.3 Role of Meeting Factor

The meeting factor influences the size of change of individuals during one meeting. The bigger the meeting factor is, the more bits are changed. When we use bigger meeting factor, the algorithm converges faster because individuals get to be similar to their friends sooner. But on the other hand, slow changes give to individuals more time for doing "good steps."

Experiments 5.1 and 5.3 showed us that a good choice of meeting factor is $p_f \in [0.1, 0.5]$ when $p_f \sim 0.5$, the algorithm is more efficient for simpler functions as for example the one-max problem and when $p_f \sim \frac{1}{8}$, the algorithm converge slower but for more complicated problems, as for example the SAT problem, it is able to converge further.

4.1.4 Role of Self Confidence

The self confidence of an individual decides how often he or she returns back to his or her original opinions when the change did not lead to a better solution. The bigger the self confidence is, the more he or she returns back.

Let us look at the self confidence from a view of exploration and exploitation. The big self confidence causes an individual to behave more reservedly and hence to be more exploitative and less explorative. On the other hand, with low self

confidence, the individual is more explorative by himself. From the population point of view when all individuals have low self confidence then they are not shame to become the same as the others so the population converge fast to a state that there is not much of new information available.

So it seems that the big self confidence is advatageous and experiments 5.1 and 5.3 support this theory. Based on them we claim that the best choice of self confidence is $p_s \sim 0.9$ or bigger. However it holds only for the case of population with a common self confidence. Populations with various self confidence open possibility of cooperation of different kinds of individuals and the whole situation become more complicated. We discuss such options and other possibilities of individuality in section 4.5.

4.2 Complexity analysis

Let us analyze the time complexity of an algorithm first. Let n be the number of individuals, m number of edges, d the size of an individual, T the total number of fitness evaluations and F the complexity of a single fitness function evaluation. During initialization we construct a graph in time $\mathcal{O}(n+m)$ and initialize individuals with their fitness values altogether in time $\mathcal{O}(nd+nF)$. In the rest of the algorithm we have bounded the number of fitness function evaluations by T and we can bound the number of evaluations of both operators by T as well since we always evaluate a fitness function after any operator call.

The mutation can be done in constant time (we just flip a single bit), and the meeting operator touches every bit of both individuals constant times so its complexity is $\mathcal{O}(d)$ if we exclude a fitness computing that can happen during the meeting. So the complexity of the running part on the algorithm is $\mathcal{O}(TF+Td+T) = \mathcal{O}(TF+Td)$. In addition, we can assume that $F \in \Omega(d)$ which is a sensible assumption that says that every bit of individual matters for fitness function. So we reduce the complexity to $\mathcal{O}(TF)$.

The complexity of the whole algorithm is $\mathcal{O}(n+m+nd+nF+TF)$. It is sensible to have $n \in o(T)$ because then we expect every individual to be a part of an interaction more than constant times, or even $m \in o(T)$ because then the expected usage of a single edge by meeting operator is more than a constant. So at the end we can estimate the whole complexity by $\mathcal{O}(TF)$. Fitness function complexity F is given by the problem to solve so the most crucial factor to pay attention to is the total number of fitness function evaluations T.

4.3 Graph Choice

The crucially important issue in a social interaction algorithm is the appropriate choice of a graph. The graph structure decides what pairs of individuals can interact with each other, and determines the distance of an individual from all others. The graph choice fundamentally influence the convergence behaviour as well as exploration and exploitation.

Two major extreme types of a graph to use are clique and circle. In a clique every pair of individuals is allowed to interact. In such case the algorithm converges very quickly. Clique is effective when solving "simple" problems as, for

example, the one-max problem and it is less effective when solving more complicated problems as, for example, the SAT problem. It converges fast but at the same time it loses a variability of individuals, and hence it has lower chance to evolve anything new.

On the other hand, circle is very sparse graph where every individual is connected just to its two neighbours. Its convergence is noticeably slower than in a case of clique. When solving simple problems like the one-max problem then using a circle seems to be a wasting of time but when solving a more sophisticated problems like the SAT problem then circle reaches much better results than any dense graph. Its convergence is slow so it has much more time to combine informations across its various population. Thanks to large average distance of individuals, there is a good variability of information obtained in genes across the population. So many parts of the search space are being discovered in parallel and there is a bigger chance of achieving a good solution.

Promising results of a circle on the SAT problem motivate us to try various types of circular graphs. During experiments we tried the following types:

- **clique** (figure 3.1)
- **circle** (figure 3.1)
- **double-circle**: the same as circle but every second vertex is connected as well (figure 4.3)
- **ladder**: two layers of a circle. It is like a real ladder but a circular one (figure 4.3) [16]
- **grid**: Standard grid on torus. Every vertex has exactly four neighbours (figure 4.3)
- **star**: Single vertex connected to $n - 1$ leafs (figure 4.3)
- **random**: A random connected graph with n vertices and $3n$ edges.

4.3.1 Clusters

We would like to construct a graph which combines advantages from both extreme graphs: the clique and the circle. That motivates us to pay attention to cluster graphs. We design a simple version of cluster graphs where we have several mutually disjoint cliques which are somehow connected by several edges. The way of connecting cliques together and the ratio of the size of clusters to the amount of clusters influences the convergence properties as we show in experiments 5.5 and 5.6.

A clique component in a cluster graph we call a *cell*. Every cell has strong convergence properties (it converges fast to some "good" solution) and at the same time memory cells exchange information between each other so altogether the cluster graph works efficiently on a global scale as well. More of interactions happen inside a cluster and less happen across two different clusters. Such behaviour might remind us an island model of genetic algorithms from section 2.7.1

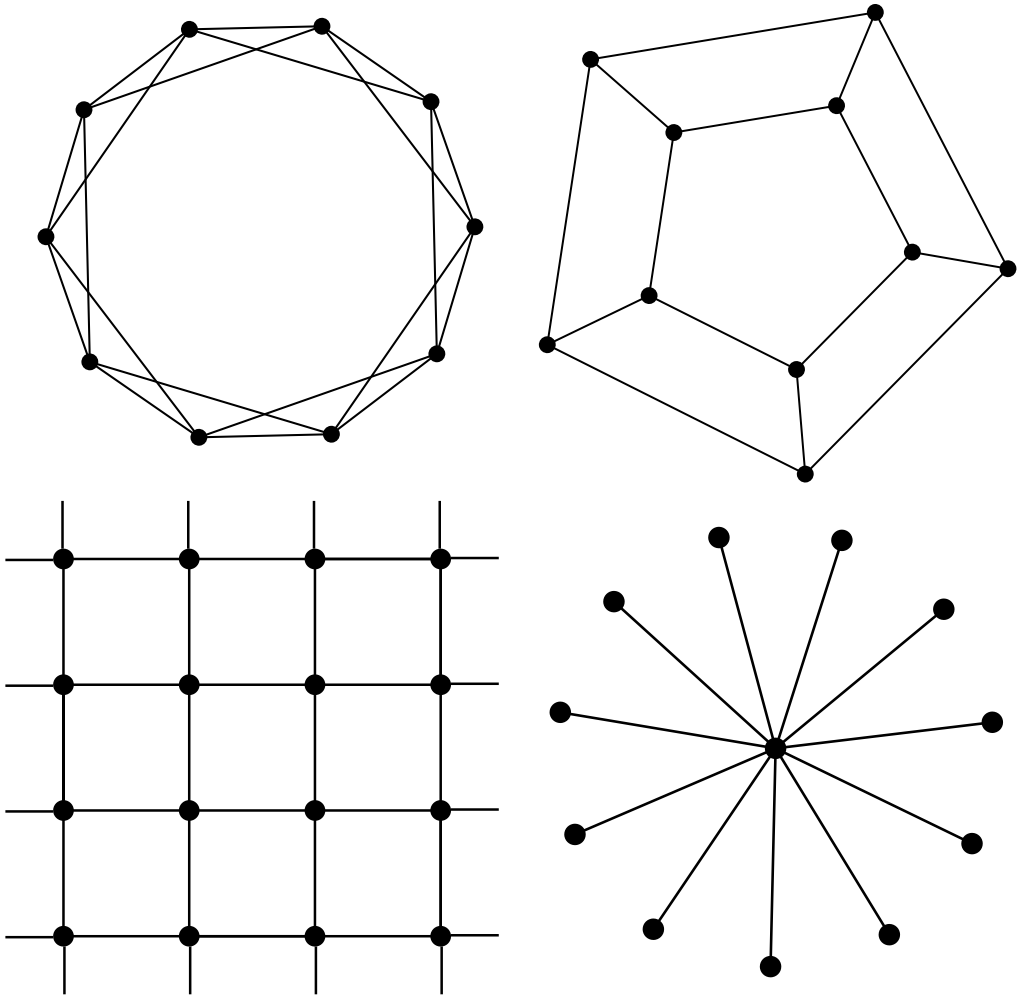


Figure 4.3: There is a double-circle on 10 vertices at the top left picture, circular ladder at the top right picture, toroid grid at the bottom left picture and a star at the bottom right picture.

and the usage of strong and weak ties in a social networks as we discussed in section 3.3.2.

We design and distinguish the following types of cluster graphs. Let us note that in every of following types all cells have the same size.

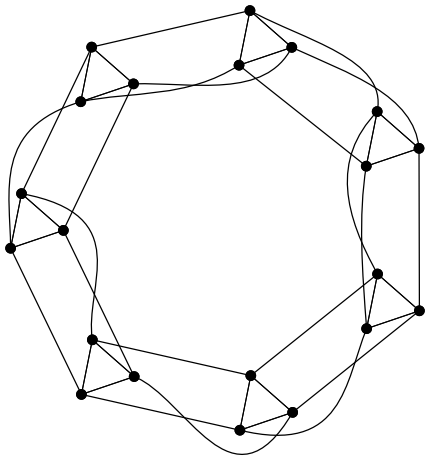
- **Cluster-circle**: we label cells by a *cell number* and vertices of every cell by *vertex number*. We make a circle on vertices of the same number in an order given by cell number. See an example on figure 4.4.
- **Cluster-clique**: we label vertices of a cell by numbers and we make a clique on every set of vertices with the same number. See example on figure 4.4.
- **Cluster-double-circle**: Every vertex of a cell independently create a new connection to a random vertex of the next cell with probability $\frac{1}{3}$, to a random vertex of the second next cell with probability $\frac{1}{3}$ and to none vertex with probability $\frac{1}{3}$. See an example on figure 4.4.
- **Random-clusters**: first we create random spanning tree on a graph of cells and then we connect every other vertex independently to a random vertex from a different cell with a probability $\frac{1}{2}$. See pseudo code on figure 4.5 for more details and an example of a graph on figure 4.4.
- **Weak-cluster-circle**: there is created just one circle on cells. Every cell has two vertices chosen. One of them is connected to the previous cell and one is connected to the next cell. A structure of this graph is the most related to the island model of the genetic algorithm described in section 2.7.1. See an example on figure 4.4.

We can see that all cluster graphs described above except cluster-clique are sparse and have a different structure of connections. In section 5.5 we compare the performance of cluster algorithms which also shows us that all sparse cluster graphs have better convergence properties than denser cluster-clique.

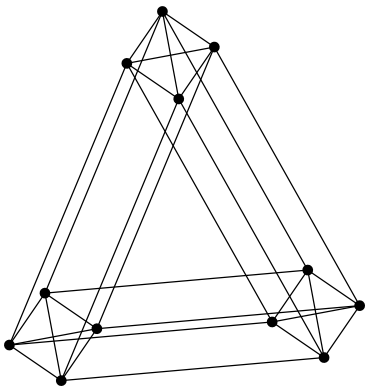
4.3.2 Usage of Tie Strength

Cluster graphs have a structure more like social networks, they combine advantages from both extreme graphs, the clique and the circle, and they have promising results. But examples in section 4.3.1 still do not have all properties of social networks. They are clustered, they are sparse random-clusters graph even has a low average distance but none of them has more weak ties than strong ties because then strong ties would not be used more than weak ties.

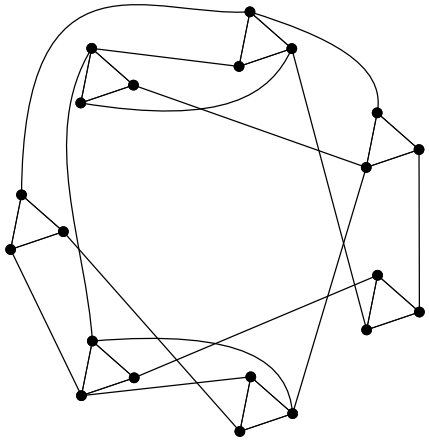
In order to have more weak ties in a graph and still use strong ties heavier than weak ties, we have to modify a social interaction algorithm to respect a tie strength when choosing an edge for the meeting operator. We follow a tie strength as formulated in a definition 9 in section 3.3. We simulate the tie strength by adding every edge to a graph repeatedly as many times as big is its tie strength. Let us note that we can always afford it because a tie strength is always a positive integer. Then the probability of an edge to be chosen is proportional to its tie strength.



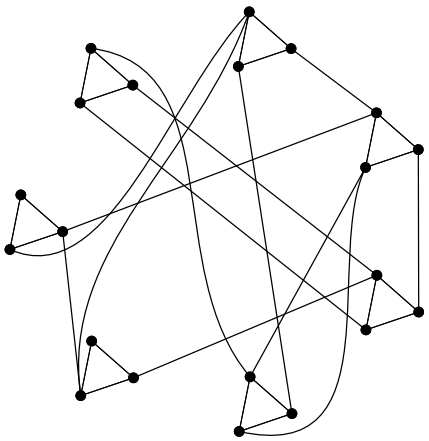
cluster-circle



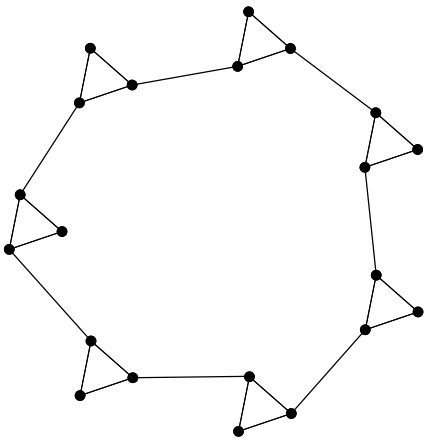
cluster-clique



cluster-doublecircle



random-clusters



weak cluster circle

Figure 4.4: Overview of cluster graphs.

Algorithm 7: Random Clusters Generator

input: The number of clusters k and the cluster size l
output: The graph of type random-clusters

```
1  G = empty graph with vertices {0, 1, ..., kl − 1};
2  for i ∈ {0, ..., k − 1} do
3  |  create a clique on vertices {il, ..., il + l − 1};
4  end
5  for i ∈ {0, ..., k − 1} do
6  |  for j ∈ {0, ..., l − 1} do
7  |  |  c = rand(k);
8  |  |  d = rand(l);
9  |  |  if with probability 0.5 and c ≠ i then
10 |  |  |  add an edge {il + j, cl + d} to G if it is not already present;
11 |  |  end
12 |  end
13 end
14 return G;
```

Figure 4.5: The pseudo code of the random clusters generator.

Introducing a tie strength in a case of a social graph cause that the social interaction algorithm behaves more like a real social network. The tie strength plays an important role especially in a case of complicated cluster graphs where clusters are not obvious. On the other hand, tie strength has no meaningful usage in case of standard graphs like clique, circle, ladder or grid because in these graphs all vertices play an equal role and a tie strength of every edge is the same.

4.4 Distinction of Social Interaction Algorithm

Recently, many new nature-inspired optimization algorithms have been developed. These algorithms are sometimes similar to each other at the end even when they take an inspiration in totally different parts of the nature. So when developing a new algorithm we must ensure that we really design something new and not just algorithm that can be easily transformed to some another already existing one. In this section we list properties which the social interaction algorithm distinct in, and we try to vindicate that there is a good reason to continue in a research of such algorithm.

- **Allow various infrastructures**: A graph structure influence the behaviour of the algorithm. A good graph structure can vary from problem to problem and based on experiments from chapter 5 we can say that various graph structures are advantageous when solving various problems.

- **No explicit selection**: There is no explicit selection in the social interaction algorithm, and individuals interact just via the graph structure. This property might seem disadvantageous for the convergence properties. On the other hand, the algorithm does not converge fast to local optima. For

example in the SAT problem there is usually no big difference in fitness value across the population so even algorithms with selection often do not have an advantage because of it.[1]

- **Individuality**: Since there is no selection, there is no need to treat every individual with the same criteria. So we have an opportunity to let every individual to behave differently and be more effective in a different part of search space. As in the real world we can take an advantage of various qualities of individuals and lead it to the success in a global scale. We explore an individuality more in the following section.

4.5 Individuality

Social interaction algorithm has no explicit selection. There is no selection which individuals can use to choose partners to interact with. They use their partners "selfishly" just to get to be better. So then there is no reason to have the same behaviour of all individuals and to treat every individual with the same criteria. We can set a different goal to every single individual, and in such way to force him or her to prefer a concrete subspace more. Or we can force some individuals to behave more explorative and others to behave more exploatative. If individuals have different goals then they can be advantageous for each other because each of them could contain a different kind of knowledge which the other one does not have. Specially, individuality can play an important role in cluster graphs. If every member of a cluster behave somehow differently then the whole cluster is more variable and has better chance to find a new good information.

In this section we introduce an individuality by unique self confidence, an individuality by unique objectives and special kind of individuals. However, an individuality theoretically has unlimited amount of options. It is just a question of creativity and opportunities of a solving problem.

4.5.1 Unique Self Confidence

Individuals can have a unique self confidence. A self confidence is a parameter that determines the size of change in case of disadvantageous meeting and hence it influences whether an individual behaves more explorative or exploatative. We designed the following two ways how to determine self confidences of individuals in population.

- **Uniform distribution**: every individual picks his or her self confidence from the interval $[a, b]$ uniformly randomly.
- **Better of two**: every individual picks two values from interval $[a, b]$ uniformly randomly and takes the bigger value as his or her self confidence. Based on experiments 5.1 and 5.3 we know that high self confidence is advantageous and leads to a better performance of the social interaction algorithm. Therefore, we are motivated to prefer higher self confidences.

[1] Let us note that for example the genetic algorithm use different approaches when solving complicated problems. It can use the tournament selection which is less dependent on the fitness values or to use some fitness scaling, et cetera.

Unique self confidence of individuals could cause better variability of an algorithm since individuals have different explorative and exploatative properties. However the experiment 5.7 did not provide any significant positive influence on the performance.

4.5.2 Personal Fitness Function

Another way of individuality is to let every individual i to slightly modify a target fitness function F and derive his or her new personal fitness function F_i from it. Then an individual uses a fitness function F_i while the best individual across the population is still chosen by the target fitness function F.

A personal fitness function opens wide range of opportunities. It can be derived generally just by preferring some bits to have concrete values, or it can targets some specifics of solved problem. It can be determined independently for every individual as well as it can make personal fitness of neighbours to be similar.

Personal fitness functions are very promising because they can bring much higher variability into the population. But we have to be careful when we prefer some parts of subspace too heavily, then we risk that an individual easily gets stuck in his or her local optimum.

Personal fitness function should mimic interests and hobbies of an individual. In the real world also not everyone wants the same and everybody has his or her own measurements of success. But externally people usually behave according to their own "fitness functions" so in the meeting operator we still use personal fitness functions even when they could not be relevant for the other individual.

We introduce two concrete examples of possible personal fitness functions, one general and one related to the SAT problem. However, there are many possibilities how to design another ones.

- **Bit value preference**: Input parameters $k \geq l$ and w. A *preference* is a pair of values (b, v). The preference (b, v) is satisfied if and only if a bit b equals to the value v. At the beginning, every individual chooses k personal preferences at random. Then every individual creates his or her final preferences by adding l random preferences from personal preferences of his or her every neighbour. Then every time a fitness function is evaluated a bonus w is added for every satisfied preference of an individual's final preferences.

- **SAT clause preference**: The SAT problem has clauses and our goal is to satisfy them all. Every individual can have his or her own weights $w_1, \ldots, w_m$ for every clauses. Different preferences then cause that each individual prefer a different subspace to converge to.

4.5.3 Special Types of Individuals

We can introduce some special kinds of individuals that have some extraordinary properties or behaviour which other individuals does not have. An individual can for example does some additional operation, or have a special structure or

to behave differently during meeting operator. Again we have many options of what to design.

As an example we try to introduce important individuals for the social epidemic arise that we described in section 3.4.

- **Connector**: An individual that has many connections. We can represent connectors as vertex that is connected to significantly more individuals than others. Optionally, we force the connector to be chosen by the meeting operator more than others but because of many incident edges he or she is chosen more often anyway.

- **Maven**: An information specialist in a given area. We choose several bits and any time a maven is modified he search a local space given by those chosen bits and takes the best solution from the subspace. Let us notice that the amount of chosen bits should be low because the size of subspace grow exponentially.

- **Salesman**: An individual who is able to influence others more than an ordinary individual. We have more sensible options how to mimic this kind of behaviour. For example, others can have a lower self confidence when meeting a salesman or others can see that a salesman has bigger fitness than he really has.

4.6 Relation to Other Models

As already mentioned, social interaction algorithm is based on genetic-like algorithms. But it is not the true population based algorithm because all individuals survive all the time and nobody is replaced by someone better from the next populations. It is close to swarm intelligence based algorithms as well but it is also not the true swarm intelligence algorithm because individuals do not explicitly work all together. We cannot say that all individuals work together to achieve a common goal like it is typical for swarm intelligence algorithms. Like in the real world individuals interact to each other and the fact that there appears some fit individual is more a corollary of a group behaviour than a public goal.

Social interaction algorithm is very rich in the amount of parameters. So now we go to discover whether it is possible to find such parameter settings that we enforce the algorithm to behave like another well-known algorithm. We prove that the social interaction algorithm can mimic the behaviour of Hill climbing method from section 2.3.

Theorem 1. *There exist parameter settings of social interaction algorithm such that it mimics an evaluation of Hill climbing method.*

Proof. Let us have the Hill climbing method that runs for T iterations, optimizes a function $f : \{0,1\}^d \to \mathbb{R}$ and changes exactly b random bits in a single step.

We use social interaction algorithm with a huge star graph where the number of leaves tends to infinity so the probability that during an algorithm we use one leaf twice for the interaction tends to zero. We can afford that, if we generate

leaves lazily.[2] We set $p_m = 0$ and $p_c = 1$ so there is no mutation happening and there is only meeting operator taking place. We set the self confidence $p_s = 1.0$ so only good steps are accepted, and bad steps are completely rejected.

The middle individual mimics the Hill climbing solution, and leafs mimics the random changes. For the middle individual we set the fitness function $F(x) = f(x)$ in the case of maximization and $F(x) = \frac{1}{1+f(x)-\min\{f(x)\}}$ in case of minimization. For individuals in leaves we set $F(x) = \infty$ in order to the middle individual be always the one who is inspired by the other. Then we must measure the algorithm performance only by the fitness of the middle individual.

The last parameter is the meeting factor. We set $p_f = \frac{2b}{d}$ because then the expected number of bits to be changed is b. In a single iteration, the leaf individual is new with a probability $p \to 1$ so his or her every bit is zero with probability 0.5 and one with probability 0.5 as well. The expected number of topics to be discussed is $d\frac{2b}{d} = 2b$ and by the linearity of expected value the expected number of changed bits is b because the probability that the leaf individual equals to middle individual on any single bit is exactly 0.5.

When we join everything together, we can say that a single evaluation of the meeting operator in the social interaction algorithm mimics a single step of the Hill climbing method and hence the social interaction algorithm with such parameters mimics the Hill climbing method. □

So, social interaction algorithm can mimic the Hill climbing method. But what about the simulated annealing which is very similar? Can we mimic it as well? The answer is: "not exactly." Simulated annealing contains a temperature T parameter and by the temperature there is decided the probability $p(T)$ of accepting a worse solution. In social interaction algorithm we can control a parameter of self confidence p_s to be $1 - p(T)$ in every single iteration. Then we obtain similar but different behaviour. The probability p in simulated annealing is applied at the whole solution at once and on the other hand the self confidence in the social interaction algorithm is applied on every single changed bit independently. So, by the social interaction algorithm we can simulate an algorithm that is very similar to simulated annealing but behave structurally differently in a case of worse solution.

Similarly, a social interaction algorithm with a clique might seem to be similar to the genetic algorithm. In both algorithms there is allowed an interaction of any pair of individuals but selection and crossover both together work fundamentally differently.

4.7 Parallelism

We can effectively run the social interaction algorithm in parallel in multiple threads and reduce the running time. We provide a lock to every individual. If a thread chooses an individual for mutation or meeting, it locks his or her and no other thread can modify him or her until the current thread is finished. But we have to be careful – in a case of the meeting operator we must lock two individuals and when a situation as in figure 4.6 appears we end up in a deadlock. This issue

[2] By generating lazily we mean that we generate a concrete leaf at the time we first need to access it and not before.

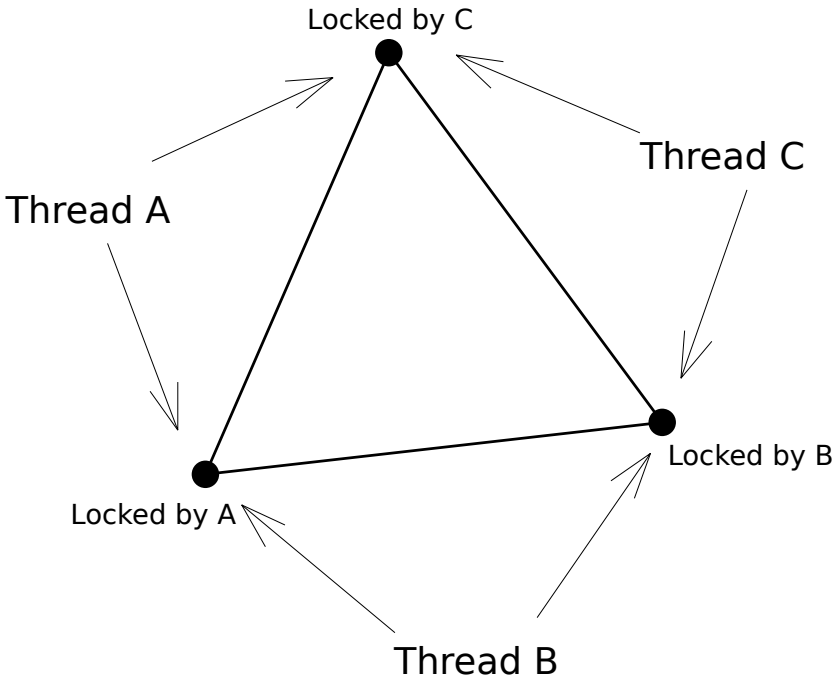


Figure 4.6: Deadlock situation. Thread A has locked the left vertex and it is waiting for the top vertex, thread B has locked the right vertex and it is waiting for the left vertex and thread C has locked the top vertex and it is waiting for the right vertex.

we can simply solve by the rule that a thread always tries to pick an individual with the lower identification number first.

Now, when we know how to parallelize the algorithm, the question is how many threads we are able to use effectively. We do not want to have many threads waiting for other threads to be done because then we are wasting the computational time by an additional thread management. We prove that when all threads together lock at most $\mathcal{O}(\sqrt{n})$ individuals at the same time then the expected number of individuals wanted by more than one thread is $\mathcal{O}(1)$. But first we need to prove the following helping lemma.

Lemma 2. *Let us have n empty queues and $\sqrt{n}$ persons. When each person choose randomly a queue to go in, then the expected number of persons waiting in any queue is lower than $\frac{1}{2}$ where by waiting we mean that a person is in the queue at least in the second position.*

Proof. Let W be a random variable that determines a number of waiting persons and T random variable that determines number of non-empty queues. Then $W = k - T$ and hence

$$\mathbb{E}[W] = k - \mathbb{E}[T]$$

Let us have n queues and k persons. Then the probability of i-th queue to be non-empty is $1 - \left(\frac{n-1}{n}\right)^k$ and by the linearity of expected value we have

$$\mathbb{E}[T] = n\left(1 - \left(\frac{n-1}{n}\right)^k\right)$$

so the expected number of waiting persons when we substitute $k = \sqrt{n}$ is

$$\mathbb{E}[W] = f(n) = \sqrt{n} - n\left(1 - \left(\frac{n-1}{n}\right)^{\sqrt{n}}\right)$$

Our goal is to show that $\forall n : f(n) < \frac{1}{2}$. It is enough to show that

i) $f(n)$ is increasing on interval $[1, \infty)$

ii) $\lim_{n\to\infty} f(n) = \frac{1}{2}$

Let us first prove that $f(n)$ is increasing on interval $[1, \infty)$. We can numerically check that $f(1) < f(2)$ so it means that if $f(n)$ is monotonous on $[1, \infty)$ then it is increasing if it is continuous (it obviously is) and has no local extremes. We check for local extremes by the first derivation.

$$f'(n) = \frac{1}{2\sqrt{n}} - 1 + \left(\frac{n-1}{n}\right)^{\sqrt{n}} + n\left(\log\frac{n-1}{n}\right)\left(\frac{n-1}{n}\right)^{\sqrt{n}}\frac{1}{2\sqrt{n}} < 0$$

on $[1, \infty)$ and hence there is no local extreme on that interval. The non-equations holds because

$$\left(\log\frac{n-1}{n}\right)\left(\frac{n-1}{n}\right)^{\sqrt{n}}\frac{1}{2\sqrt{n}} < 0$$

and it holds because $\log\frac{n-1}{n} < 0$ and other multipliers are positive since $n \geq 1$. The rest of the expression of $f'(n)$ is

$$\frac{1}{2\sqrt{n}} - 1 + \left(\frac{n-1}{n}\right)^{\sqrt{n}} \leq 0$$
$$1 \leq 2\sqrt{n}\left(1 - \left(\frac{n-1}{n}\right)^{\sqrt{n}}\right)$$

The last inequality holds because $\left(\frac{n-1}{n}\right)^{\sqrt{n}} \in [0, 1)$ and hence the right side is at least 1 since $n \geq 1$. So we have proven that $f(n)$ is increasing on $[1, \infty)$.

Now let us compute the limit. We do it mechanically.

$$\lim_{n\to\infty}\left[\sqrt{n}-n\left(1-\left(\frac{n-1}{n}\right)^{\sqrt{n}}\right)\right]=$$
$$\lim_{m\to\infty}\left[m-m^2\left(1-\left(\frac{m^2-1}{m^2}\right)^{m}\right)\right]=$$
$$\lim_{x\to 0}\left[\frac{1}{x}-\frac{1}{x^2}\left(1-\left(\frac{\frac{1}{x^2}-1}{\frac{1}{x^2}}\right)^{\frac{1}{x}}\right)\right]=$$
$$\lim_{x\to 0}\frac{x-1+(1-x^2)^{\frac{1}{x}}}{x^2}=$$
$$\lim_{x\to 0}\frac{1+(1-x^2)^{\frac{1}{x}}\left(\frac{-2}{1-x^2}-\frac{\log(1-x^2)}{x^2}\right)}{2x}=$$
$$\lim_{x\to 0}\frac{1}{2}\left[A+B+C+D\right]$$

where

$$A=(1-x^2)^{\frac{1}{x}}\left(\frac{-2}{1-x^2}-\frac{\log(1-x^2)}{x^2}\right)\frac{-2}{1-x^2}$$
$$B=(1-x^2)^{\frac{1}{x}}\frac{4x}{(1-x^2)^2}$$
$$C=(1-x^2)^{\frac{1}{x}}\left(\frac{-2}{1-x^2}-\frac{\log(1-x^2)}{x^2}\right)\frac{\log(1-x^2)}{x^2}$$
$$D=-(1-x^2)^{\frac{1}{x}}\frac{\frac{x^2}{1-x^2}-2x\log(1-x^2)}{x^4}$$

In the first equation we use a substitution $m=\sqrt{n}$, in the second equation we use a substitution $x=\frac{1}{n}$ and in the fourth and the fifth equation we apply L'Hospital rule. When we compute every addend and multiplier as separate limit, so we get

$$\lim_{n\to\infty}f(n)=\frac{1}{2}[1\cdot(-2+1)\cdot(-2)+1\cdot 0-1\cdot(-2+1)\cdot(-1)-1\cdot 0\cdot 0]=\frac{1}{2}$$

We promise that all assumptions in all L'Hospital rules are satisfied and we afford to omit the detailed description and another exhausted mechanical computations. □

Let us note that if we would have $\sqrt{cn}$ people in queues instead $\sqrt{n}$ then the result of the limit would be $\frac{c}{2}$.

By the application of lemma 2 we obtain the following proposition.

Proposition 3. *Let us assume a running of the social interaction algorithm using a r-regular graph G with n vertices. If we run the algorithm in parallel with usage of $\mathcal{O}(\sqrt{n})$ threads then the expected number of waiting threads at any time of the algorithm is $\mathcal{O}(1)$.*

Proof. A graph G is r-regular so it has exactly $\frac{nr}{2}$ edges. Let us consider the social interaction algorithm with a graph G that is running in $c\sqrt{n}$ threads for some positive c. For simplicity we consider only usage of the meeting operator because it consumes more resources than the mutation.

When a thread processes the meeting operator at a time, it blocks exactly $2r - 1$ edges from being used by other threads which is at most $\frac{4}{n}$ of all edges. A thread is in waiting state if its random picked resources (means an edge) is already blocked by another thread. At any time of the algorithm, every thread has picked a random edge from a graph and we are interested in expected number of edges that shares resources with other already processed edge. Since edges are picked randomly with the equal probability we can apply lemma 2 for $\frac{n}{4}$ queues and $c\sqrt{n}$ persons. So, for $c = \frac{1}{2}$ the expected number of waiting threads is $\frac{1}{2}$ and for any other positive c the expected number of waiting threads is $\frac{(c-0.5)^2}{2}$ which is still a constant. □

The proposition 3 holds for regular graphs like circle, clique, grid, ladder and regular clusters. We are going to extend it to the theorem 4 that holds for any general graph.

Theorem 4. *Let us assume a running of the social interaction algorithm with a graph G with m edges and the maximum degree Δ. If we run the algorithm in parallel with usage of $\mathcal{O}(\sqrt{\frac{m}{\Delta}})$ threads then the expected number of waiting threads at any time of the algorithm is $\mathcal{O}(1)$.*

Proof. In the worst case we have all vertices with a degree Δ so we have $n = \frac{m}{2\Delta}$ vertices. Now we simply apply the proposition 3 and we obtain the result. □

Chapter 5

Experiments

In this chapter we present experiments exploring the performance and properties of the social interaction algorithm. Our main goals are to find a good parameters setting, to examine an influence of its various features. We also compare the social interaction algorithm to other existing models.

We use two optimization problems in the experiments: the one-max problem and the SAT problem, both introduced in section 2.8. We use the one-max problem mostly for initial parameter tuning and to observe convergence properties, and we use the SAT problem for initial parameter tuning as well but then we focus on it more in other experiments.

We are interested in two resulting properties. Primarily we are interested in the convergence properties which we measure by the best individual evolved since the algorithm started and secondarily we are interested in genes variability [16]. We explain the purpose of genes variability in the following section.

5.0.1 Genes Variability

A *genes variability* means in how many bits individuals differ in average across the population. The bigger the genes variability is, the higher is a chance that the algorithm finds out something new. When a variability tends to zero then all individuals are moreless the same and there cannot be explored anything new by individual's interactions.

We use two types of variabilities: *local* and *global*. In the local variability we consider all pairs of individuals sharing an edge, and in global variability we compute the variability for the whole population (the same as local variability for a clique). The local variability tells us how likely the algorithm finds a new solution in the near future and omits a variability of not directly connected pairs of individuals. On the other hand, the global variability considers all the variability in the population and tells us how likely the algorithm finds a new solution in arbitrarily distant future.[1]

[1]The variability is contained somewhere in the graph but individuals with different genes information could be far from each other. So, it might last longer before the variability meets, if ever. For circular graphs it does not make a good sense to compute the global variability because the average distance of individuals is high. On the other hand, the average distance in clustered graphs, especially in random clusters, is small, so the global variability makes a better sense.

Processor	8 x Intel® Core™ i7-3700 CPU
Frequency	3.40GHz
Number of cores	8
Memory	15.5GiB of RAM

Table 5.1: Machine parameters

Algorithm	algorithm we use in the experiment
Runs	how many independent runs we execute (usually 25)
Iterations	maximum fitness function evaluations during single run
Problem instance	problem instance (or list) we run the experiment with (includes the size of individuals)
Fitness function	what fitness function we use
Population	population size
Graph type	list of graphs we use
Self confidence	self confidence setting
Meeting factor	meeting factor setting
Variability	local / global / none
Additional notes	additional notes if any
Running time	Total running time of all parameter settings

Table 5.2: Sample experiment parameters specifications.

It is time consuming to compute a local variability for denser graphs, so we use a global variability more often.

5.0.2 Machine Parameters

As we noticed in section 4.2, we are interested in complexity mostly by number of fitness function evaluations until we reach a given precision. However, a real running time is also important and we present it together with results as well. We run all the experiments on computer with parameters described in table 5.1.

5.0.3 Experiment Parameters

For all experiments we set mutation probability $p_m = 0.1$ and meeting probability $p_c = 0.9$. We describe other parameters of the experiment by the table as is the sample table 5.0.3 that contains an explanation of every paramater we use.

We may use a list of values for some parameters. Then it means that we run the experiment for all combinations of parameter settings given by all lists.

5.0.4 Results Understanding

To demonstrate results we use graphs of convergence, graphs of variability convergence, and summary tables of algorithm convergence. In graphs and tables, we always visualise the distance from the maximum possible fitness value rather than the fitness value itself. So, we can always read the data as we do the minimization with the optimum value in zero.

We repeat all experiments in 25 independent runs, and we log the worse solution, first quartile, median, third quartile, and the best solution from those 25 runs regularly after some number of iterations. Then,we show the following in the graphs: we plot the median progression by the dark line, around it we plot a semi-dark strip between the first and third quartiles progressions, and a light strip between the worse and the best solution progressions.

We plot the convergence and the variability convergence in the same way. Let us note that in the case of convergence we want to have as fast progression as possible while converging as far as possible (in ideal case into zero). In case of variability convergence we would like to converge slower because while the genes variability is high, the algorithm has a better chance to find something new.

In tables, we show convergence properties as well. In the leftmost column there are fitness values. All other columns belong to median or best solution, and contain the iteration number when we first reached a given fitness value for a given column.

Besides convergence tables, we use also summary tables. We use summary tables in experiments where we are interested only in final results. For every parameter settings (given by the table) we provide median solution, the best solution, and the variance of all solutions.

5.1 One-Max Problem Performance

5.1.1 Motivations and Goals

The goal of this first experiment is to examine convergence properties of the social interaction algorithm for various parameter settings. We want to observe how the self confidence p_s, the meeting factor p_f, and the graph choice influence the algorithm behaviour when solving the one-max problem.

5.1.2 Experiment Description

We run the experiment with all combinations of parameters listed in table 5.3. We use grid of size $\sqrt{n} \times \sqrt{n}$ and random-clusters with $\sqrt{n}$ clusters of size $\sqrt{n}$, where n is the population as well as the graph size.

5.1.3 Results

First we look at how the self confidence and the meeting factor influence a convergence properties. For all graphs, population sizes, individual sizes, and a fixed meeting factor it holds that the bigger the self confidence is the faster the convergence is and the better the final result is.

About the meeting factor, the situation is more curious. For big populations there holds that the bigger the meeting factor is, the better the convergence is. That is probably caused by better genes variability in the bigger initial population. On the other hand, for small populations and low meeting factors, the algorithm converges further than in a case of high meeting factors. That is caused probably by slower variability convergence in case of low meeting factors.

Algorithm	social interaction algorithm
Runs	25
Iterations	500,000
Problem instance	one-max problem with individual sizes 64, 1024, 2048, 4096
Fitness function	a sum of bits
Population	49, 100, 400, 900
Graph type	clique, circle, double-circle, ladder, grid, star, random, random-clusters
Self confidence	0.3, 0.5, 0.7, 0.9
Meeting factor	$\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$
Variability	local for all graphs except clique, random, and random-clusters for which we used global
Running time	132 hours, 13 minutes and 15 seconds

Table 5.3: Description of parameters of one-max problem performance experiment.

See figures 5.1 and 5.2 to check the convergence properties with clique on 400 vertices and clique with 49 vertices. See figures 5.3 and 5.4 to check the variability convergence on the same graphs. Based on these results we can say that lower meeting factors help to smaller populations to use information in their genes better for the price of slower convergence.

In tables 5.4 and 5.5 we can see a comparison of convergence properties of various graphs on $n = 100$ vertices and meeting factors $\frac{1}{2}, \frac{1}{8}$. It is obvious that for meeting factor $\frac{1}{2}$ the circular graphs are able to converge further, and for lower meeting factors like $\frac{1}{8}$ this property disappears, and denser graphs converge faster. But in both cases we can observe that lower meeting factors are significantly better for the further convergence.

5.2 One-Max Problem – Model Comparison

5.2.1 Motivations and Goals

In the experiment 5.1 we have found out some good parameter settings to solve the one-max problem by using the social interaction algorithm. Now we want to compare its performance to other existing models. We compare both, convergence properties and genes variability, to the genetic algorithm and the neighbourhood model.

5.2.2 Experiment Description

We run the experiment with all combinations of parameters as described in the table 5.2.2. In the case of the genetic algorithm we use no graph, self confidence, or meeting factor, and in the case of the neighbourhood model we use no self confidence, or meeting factor.

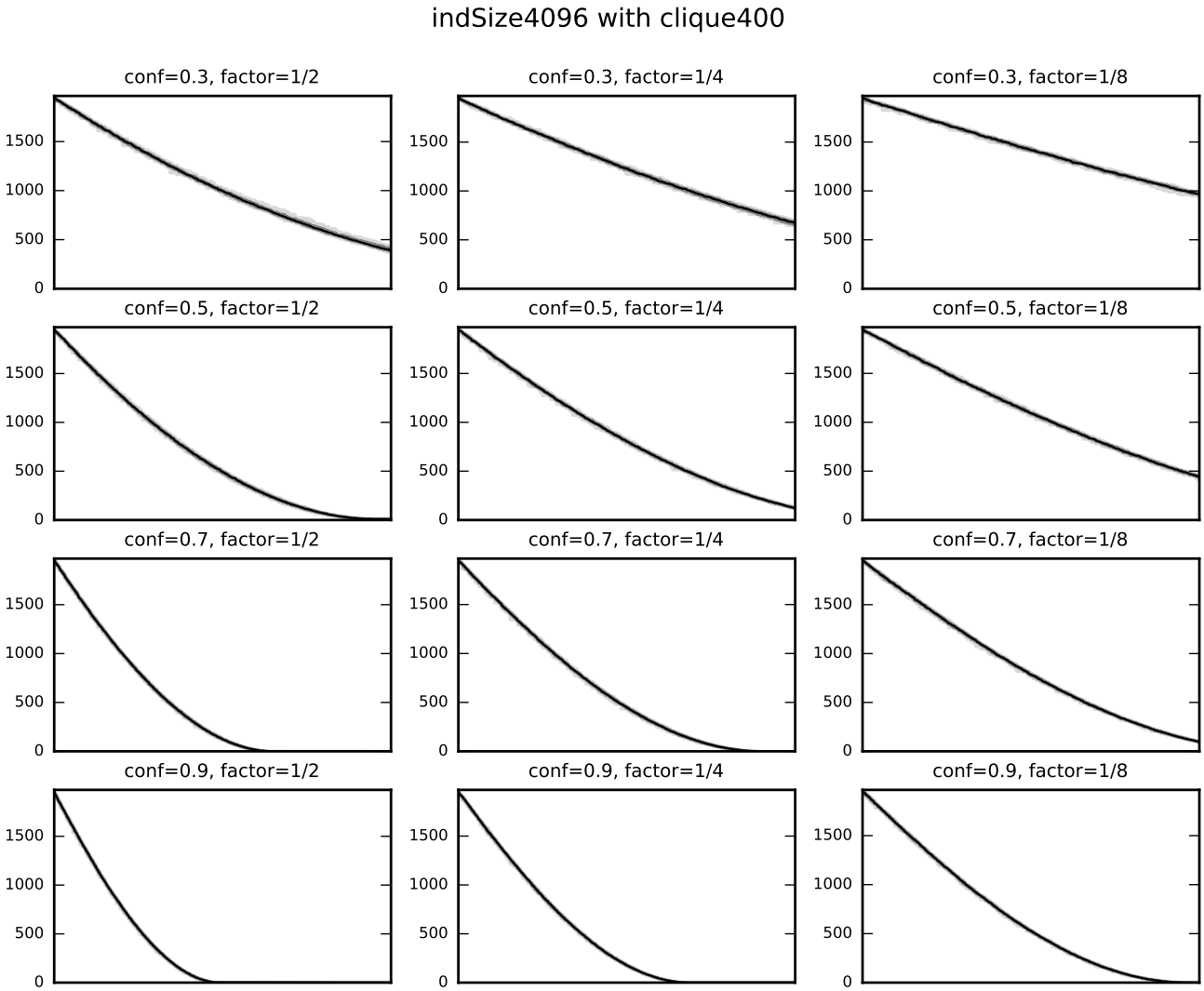


Figure 5.1: Convergence properties of social interaction algorithm when solving one-max problem with an individual size 4096 and a clique on 400 vertices. We can see that the bigger the self confidence is the faster the convergence is, and the bigger the meeting factor is, the faster the convergence is.

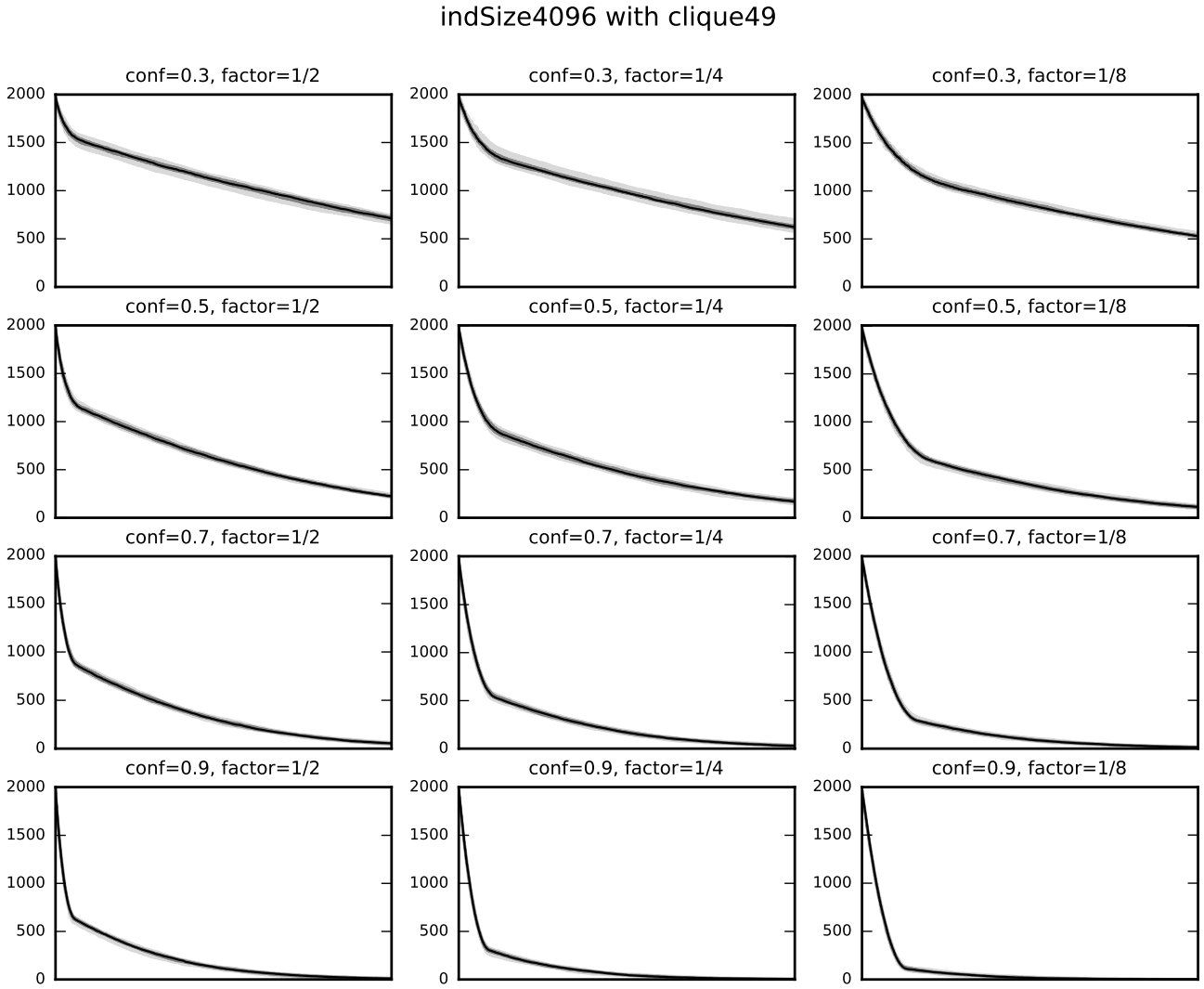


Figure 5.2: Convergence properties of social interaction algorithm when solving one-max problem with an individual size 4096 and a clique on 49 vertices. We can see that the bigger the self confidence is the faster the convergence is, and the lower the meeting factor is the slower the convergence is at the beginning but more efficient altogether.

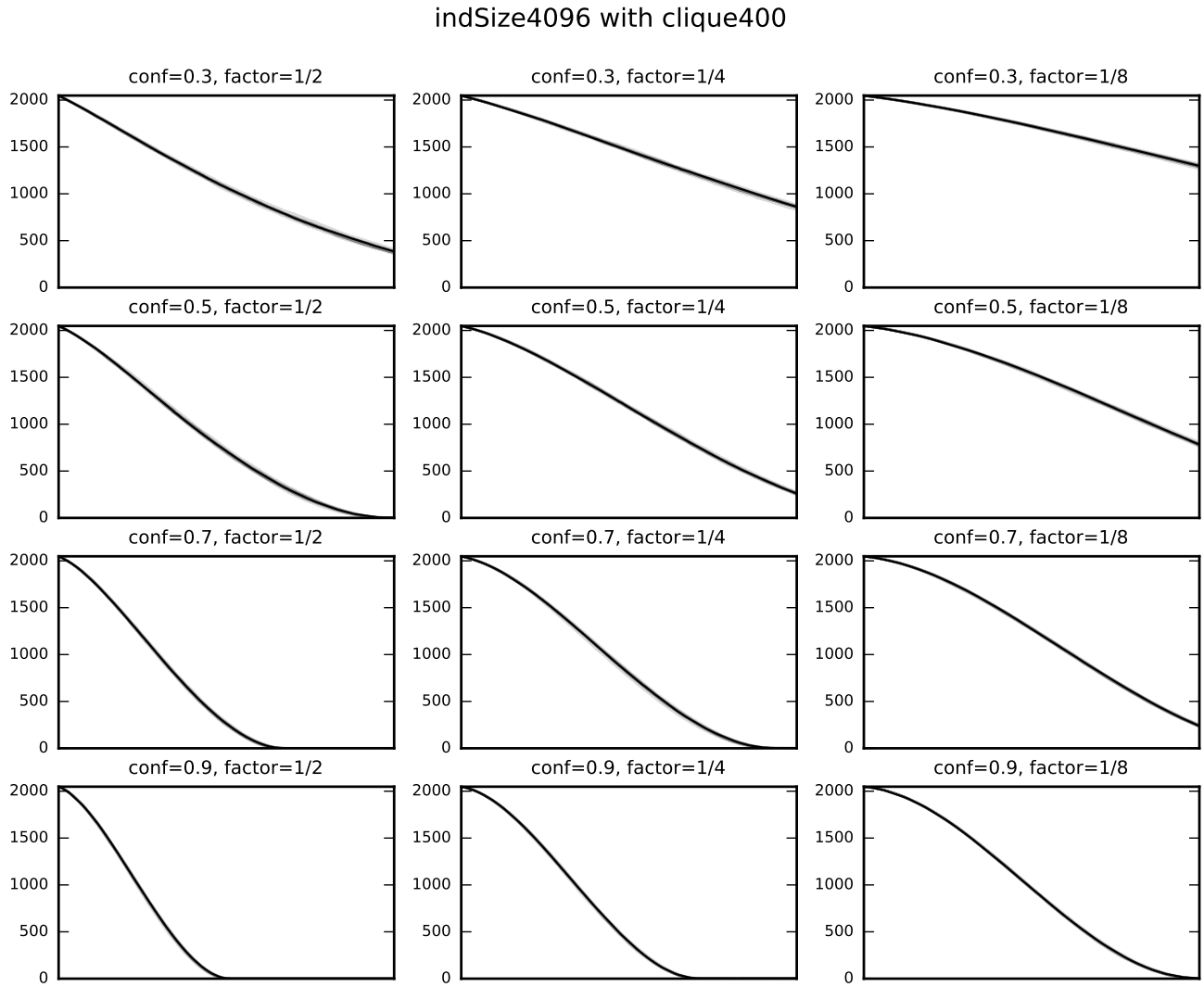


Figure 5.3: Genes variability convergence of social interaction algorithm when solving one-max problem with an individual size 4096 and a clique on 400 vertices. Naturally the convergence increases directly proportional to both the self confidence and the meeting factor.

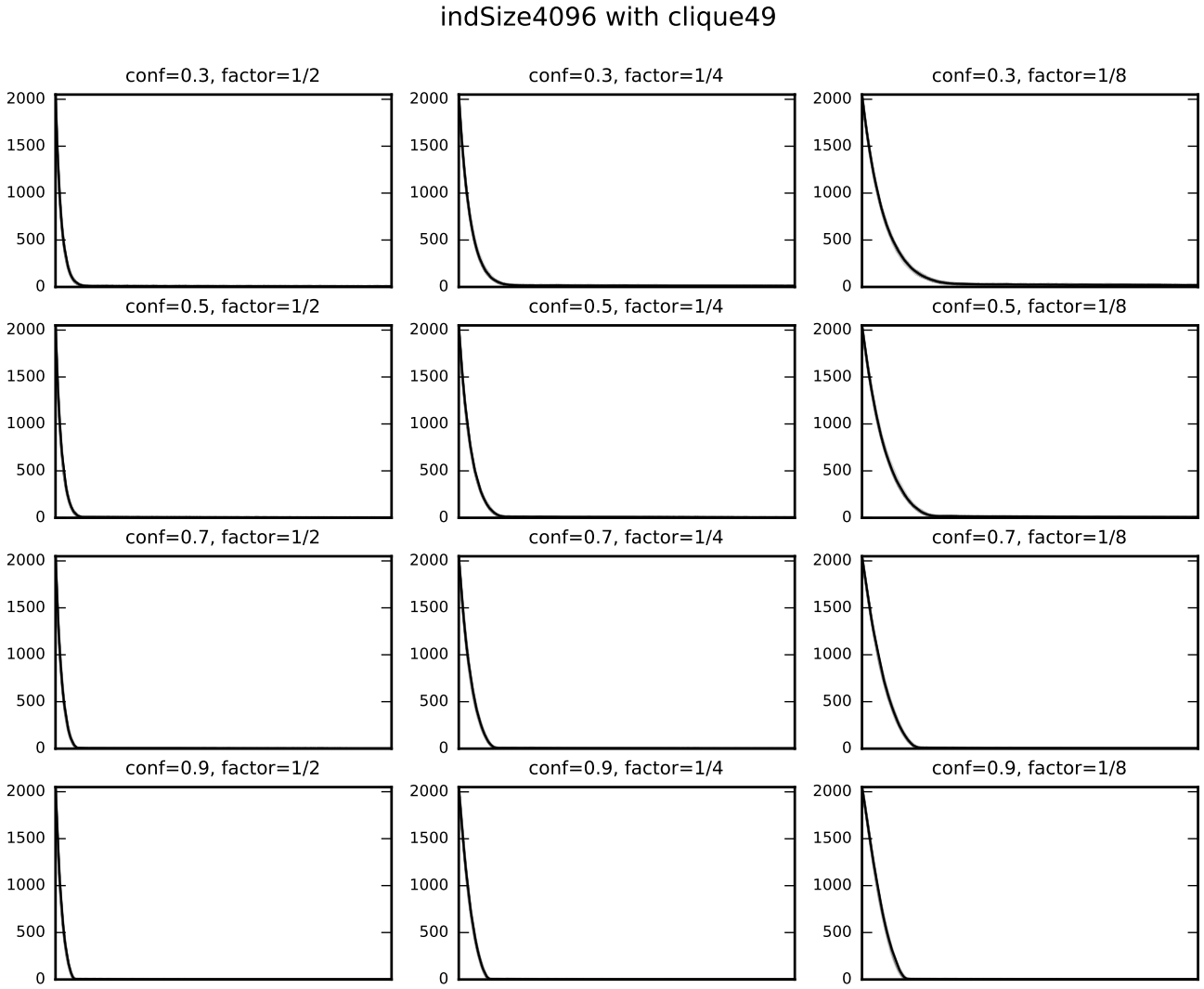


Figure 5.4: Genes variability convergence of social interaction algorithm when solving one-max problem with an individual size 4096 and a clique on 49 vertices. Naturally the convergence increases directly proportional to both the self confidence and the meeting factor. See that the convergence on figure 5.2 rapidly slows down when genes variability reaches zero.

	Clique		Circle		Double-circle		Ladder	
Fitness	median	best	median	best	median	best	median	best
1024	21401	20201	59301	56600	42800	41301	42601	40301
512	38901	37301	112701	110100	82000	79101	81401	79300
256	53100	51800	151101	147901	109201	106801	107801	106300
128	96900	82100	176601	174301	128300	124901	126600	123700
64	174500	154600	194800	189801	142401	137700	140401	138201
32	246600	220700	208001	202901	152901	146800	150700	147300
16	322400	282100	216500	210900	161200	154500	160500	153901
8	391300	318400	223900	216600	189900	159100	200400	162500
4	450100	372100	231100	224200	262300	197100	282900	192400
2	X	445300	238600	226200	339700	211700	342500	234000
1	X	463600	260500	228300	386000	223600	401000	246500
0	X	474000	324100	235100	472700	299500	451400	311500

	Grid		Random		Random-clusters		Star	
Fitness	median	best	median	best	median	best	median	best
1024	27501	26401	22401	21301	28101	25000	18401	17101
512	49301	47601	41500	40501	50901	48101	96500	78901
256	64701	62700	62601	58500	68500	64801	219601	205300
128	77201	74301	154900	136300	83601	78900	322900	292600
64	106901	87101	241900	211100	145400	109301	407800	390900
32	184300	155800	320400	287100	222500	188101	481000	453900
16	255600	215500	395900	332200	280200	251200	X	X
8	321000	264900	447500	410300	353100	302300	X	X
4	384900	290800	X	463800	404200	341800	X	X
2	457000	328700	X	477300	464400	349600	X	X
1	495400	334100	X	497000	X	356800	X	X
0	X	433000	X	X	X	X	X	X

Table 5.4: Overview of convergence properties of various graphs for $n = 100$, the self confidence $p_s = 0.9$, and the meeting factor $p_f = \frac{1}{2}$. We can see that more connected graphs like clique, grid, or random graph are faster at the beginning, and on the other hand, circular graphs like circle, double-circle, and ladder are slower at the beginning but converge further.

	Clique		Circle		Double-circle		Ladder	
Fitness	median	best	median	best	median	best	median	best
1024	39701	37601	83601	78601	63501	61601	63401	61801
512	67901	66301	159401	155701	120000	114201	118801	116300
256	87201	85101	213000	209001	160001	157501	159000	152601
128	100200	98501	250901	246801	188100	184801	186000	179601
64	109501	107501	277300	270100	207200	203100	204901	196901
32	116100	113801	296500	288200	221700	217200	218701	210600
16	120901	118601	310200	302600	232200	227201	229400	221101
8	124700	121400	321801	312801	240300	235600	236800	231400
4	134500	123601	331100	323700	245300	239500	243100	235900
2	164200	125400	337200	328000	250000	243100	246500	237300
1	203100	125800	342300	334200	253400	247100	249100	239300
0	285000	127400	350500	338700	258000	252100	253800	245300

	Grid		Random		Random-clusters		Star	
Fitness	median	best	median	best	median	best	median	best
1024	49100	46901	38901	35601	47901	44601	27900	24900
512	84700	82201	67501	65101	84300	81201	51501	49800
256	108600	106500	87801	84501	109900	104501	70601	67801
128	124901	122400	101800	97901	126801	122700	89001	84901
64	136301	133301	111300	108000	138801	134001	157700	129400
32	143501	140801	119001	114601	146800	142301	233500	207100
16	148801	147001	124601	120501	152301	147901	311200	263800
8	152900	150900	146000	125301	157100	151901	375200	314900
4	155600	153700	194300	131500	160600	154701	435900	366700
2	157700	154201	254500	134900	165700	157801	497400	416500
1	159900	156500	327500	136000	166901	161100	X	441600
0	230100	158700	359400	166100	217700	163300	X	465000

Table 5.5: Overview of convergence properties of various graphs for $n = 100$, the self confidence $p_s = 0.9$, and the meeting factor $p_f = \frac{1}{8}$. We can see that for such settings the convergence is more effective for more connected graphs like clique, or grid than circular graphs like circle, double-circle, or ladder.

Algorithm	social interaction algorithm, genetic algorithm and neighbourhood model
Runs	25
Iterations	500,000
Problem instance	one-max problem with an individual sizes 64, 1024, 2048, 4096
Fitness function	a sum of bits
Population	49, 100, 400, 900
Graph type	clique, circle, ladder, grid, random-clusters
Self confidence	0.9
Meeting factor	$\frac{1}{2}, \frac{1}{8}$
Variability	local for sparse graphs and global for dense graphs and genetic algorithm
Running time	12 hours, 34 minutes and 7 seconds

Table 5.6: Description of parameters of one-max problem model comparison experiment.

5.2.3 Results

First we look at the performance when all algorithms run in the most common conditions. It means that we choose clique for both algorithms because it is the most similar to the genetic algorithm, and we set the meeting factor $p_f = \frac{1}{2}$. We can see the results for population sizes 49 and 400 in figure 5.5. Then we use better parameters for each of algorithms independently, and compare them on population of a size $n = 100$ individuals. For the social interaction algorithm we use random-clusters and a meeting factor $p_f = \frac{1}{8}$, for neighbourhood algorithm we use grid (it was originally developed with grid and ladder [16]), and in genetic algorithm there are no additional parameters to set. See the results in figure 5.6 and in more details in table 5.7.

Based on figures and the table 5.7 we can conclude that the social interaction algorithm has slower convergence at the beginning but it is more efficient in the finishing phase. It is probably because of the slower genes variability convergence and the difference between meeting operator and standard crossover operator.

5.3 SAT Problem Performance

5.3.1 Motivations and Goals

The goal of this experiment is to research convergence properties and genes variability when solving the SAT problem, and to find the best parameters and the best types of graphs to use. We also check whether the results correspond to results of experiment 5.3 where we do the same with the one-max problem.

5.3.2 Experiment Description

We run the experiment with all combination of parameters described in table 5.8. We use two different SAT problem instances. The first is a randomly gen-

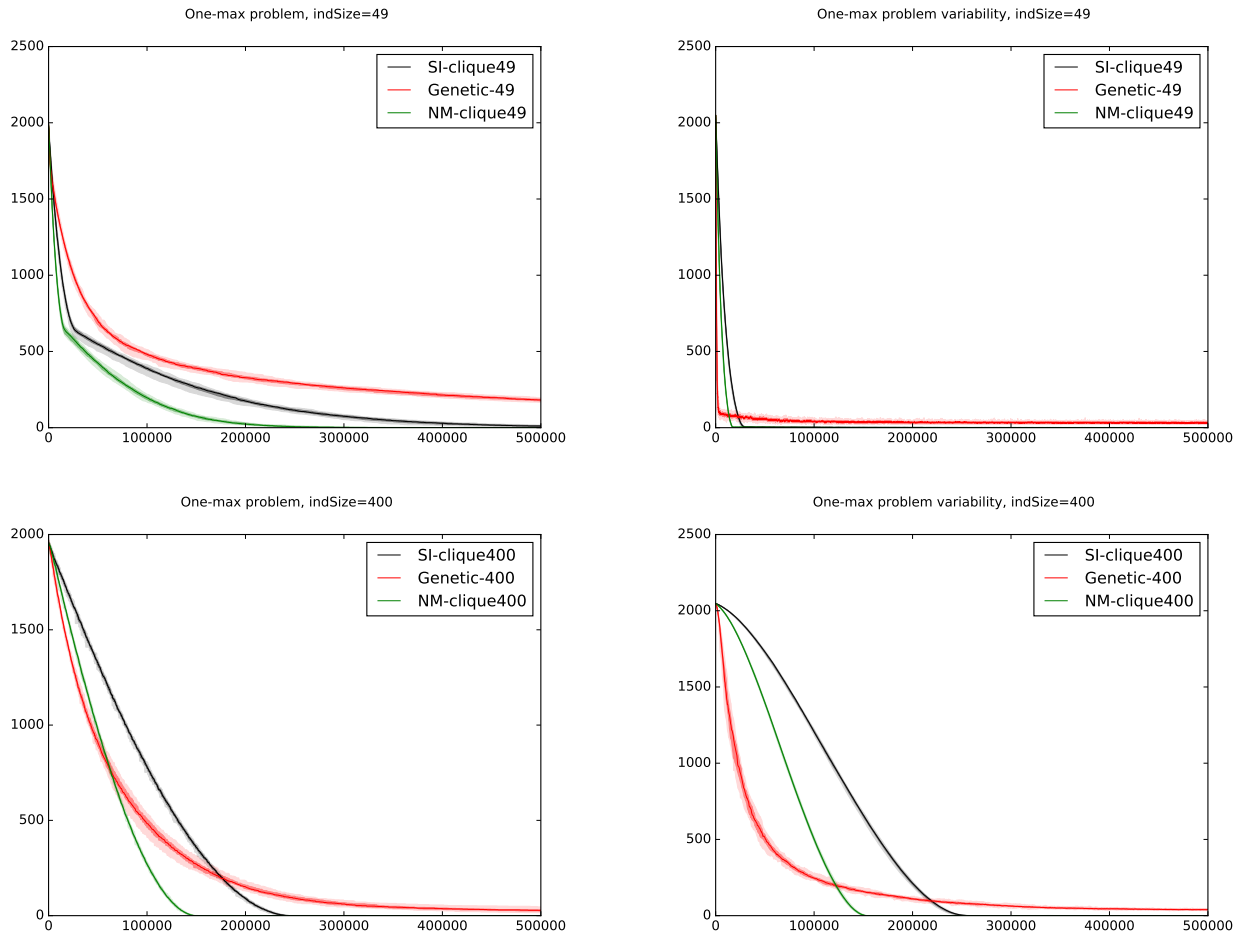


Figure 5.5: Model comparison when we use a clique. We can see the convergence progress in the left column and the genes variability convergence in the right column. It seems that in case of clique the neighbourhood algorithm is the best of all. The convergence of the genetic algorithm is fast at the beginning but it has problems to converge completely. The social interaction algorithm's convergence is slower so is the convergence of genes variability.

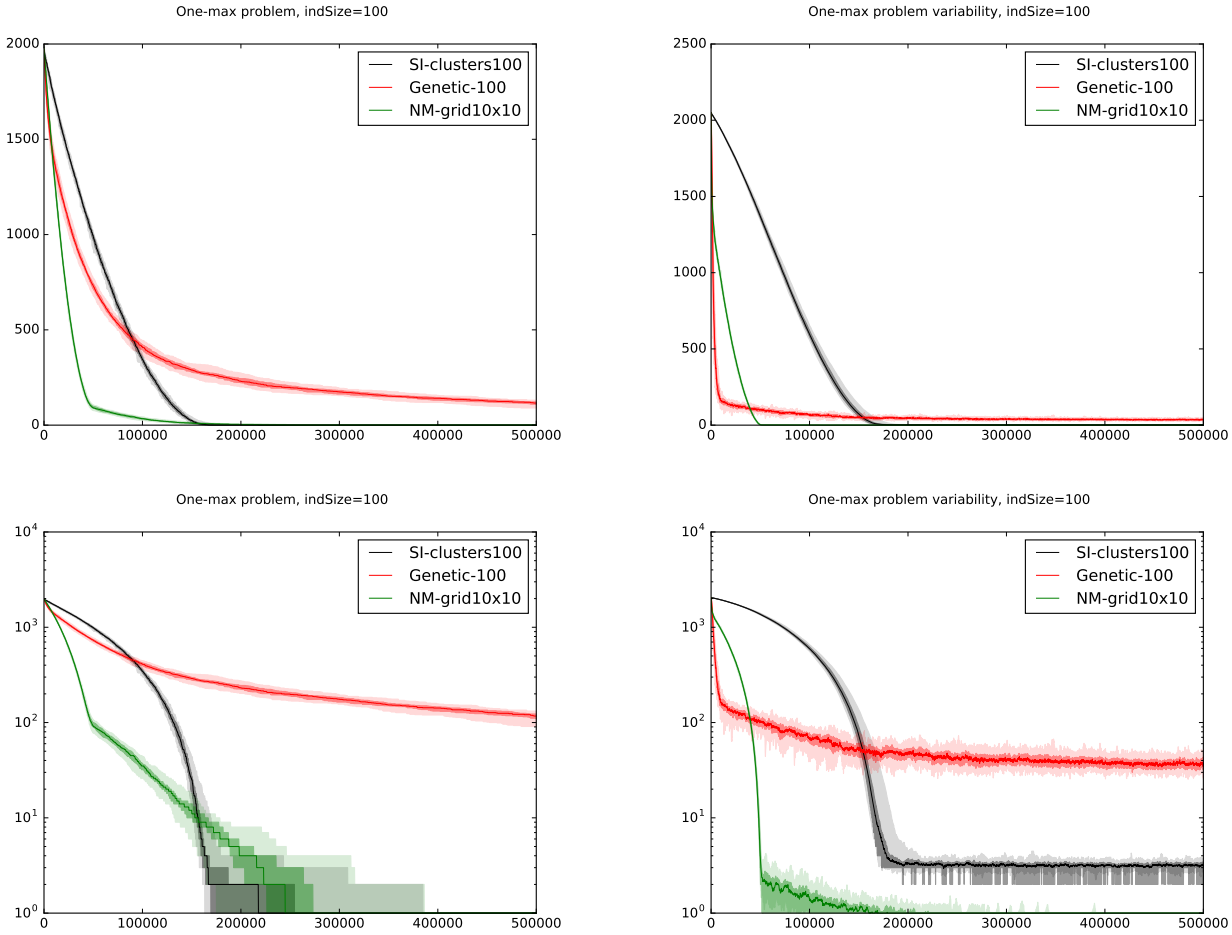


Figure 5.6: More specialized algorithms with population size 100. We can see similar results as in figure 5.5. In the first row there is linear scaled visualization, and in the second row there is a log scale of y-axis.(In this case all values are increased by one because log(0) is not defined, and log(1) = 0.) In addition, in table 5.7 we can see detailed convergence when the objective is approaching the optimum. The result is that the social interaction algorithm is the most effective in the finishing phase but it is slower at the beginning. It is probably caused by the slow genes variability convergence.

	SI-clusters		NM-grid		Genetic	
Fitness	median	best	median	best	median	best
1024	47901	44601	15800	15600	28000	24200
512	84300	81201	28000	27000	78400	73300
256	109900	104501	36800	36000	177700	146500
128	126801	122700	44400	43000	446200	352600
64	138801	134001	69600	60000	X	X
32	146800	142301	102600	88200	X	X
16	152301	147901	131000	114600	X	X
8	157100	151901	156800	133400	X	X
4	160600	154701	187800	151000	X	X
2	165700	157801	216000	162600	X	X
1	166901	161100	223200	165400	X	X
0	217700	163300	245000	175200	X	X

Table 5.7: More detailed description of convergence that we can see in figure 5.6. We can see that the social interaction algorithm is the most efficient in the finishing phase but it is slower at the beginning. It is probably because of the slow genes variability convergence.

erated 3-SAT with 125 variables and 500 clauses, since it is randomly generated it probably has more different optimal solutions. The second one is the SAT formula based on number factorization. It has 118 variables and 548 clauses and it is more tough to solve.

5.3.3 Results

First let us look at results of random SAT instance with 125 variables and 500 clauses. In figures 5.7 and 5.8 we can see the convergence properties when using clique with 400, or 49 vertices respectively. In figures 5.9, and 5.10 we can see genes variability properties respectively.

An important issue, we can observe and was not noticeable in case of the one-max problem, is the relation between meeting factor p_f and the genes variability convergence properties. The lower the meeting factor is the higher is the value that the genes variability converges to. That is an advantageous property because the algorithm has a bigger variability across the population, and hence it has a better chance to find something new.

In tables 5.9 and 5.10 we can check results for various graphs with $n = 100$ vertices, and the meeting factors $\frac{1}{2}$ and $\frac{1}{8}$ respectively. We can see that the meeting factor $\frac{1}{8}$ is not obligatory when comparing to the meeting factor $\frac{1}{2}$ as it is in case of the one-max problem. The random SAT instance is not that tough as other SAT formulas based on real problems. Such formulas typically have only few optimal solutions. So, let us present the results of the SAT formula based on factorization with 118 variables and 548 clauses in table 5.11. We can see there bigger differences than in case of the random SAT instance.

We can conclude that lower meeting factors converge slower but are able to converge further than higher meeting factors. We can also see that in case of factorization problem our solution is often not perfect. It is caused primary because

Algorithm	social interaction algorithm
Runs	25
Iterations	500,000
Problem instance	SAT problem with instances random125-500 and factor118-548
Fitness function	SAT fitness function
Population	49, 100, 400, 900
Graph type	clique, circle, double-circle, ladder, grid, star, random, random-clusters
Self confidence	0.3, 0.5, 0.7, 0.9
Meeting factor	$\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$
Variability	local for all graphs except clique, random and random-clusters for which we used global
Running time	25 hours, 33 minutes and 20 seconds

Table 5.8: Description of parameters of the SAT problem performance experiment.

the algorithm does not converge perfectly in 500,000 iterations, and secondary because parameter settings is not perfect yet. Hence we are motivated to find better graphs and algorithm features to achieve the better performance.

5.4 SAT Problem – Model Comparison

5.4.1 Motivations and Goals

In the experiment 5.3 we have found some good parameter settings to solve the SAT problem by the social interaction algorithm. Now we want to compare its performance to other existing models. We compare both, convergence properties and genes variability, to the genetic algorithm and the neighbourhood model.

5.4.2 Experiment Description

We run the experiment with all combinations of parameters described in table 5.12. In case of the genetic algorithm we use no graph, self confidence, or meeting factor, and in case of the neighbourhood model we use no self confidence, or meeting factor. We use the same SAT problem instances as described in experiment 5.3.

5.4.3 Results

First we compare different models in similar conditions. It means we use a clique in the social interaction algorithm and in the neighbourhood model because cause the most similar behaviour to the genetic algorithm. We set meeting factor $p_f = \frac{1}{2}$ because then the meeting operator mimics the standard crossover the most. We compare models on the random SAT problem instance with 125 variables and 500 clauses, and population sizes 49 and 400. There is no significant difference in convergence. Check results in figure 5.11.

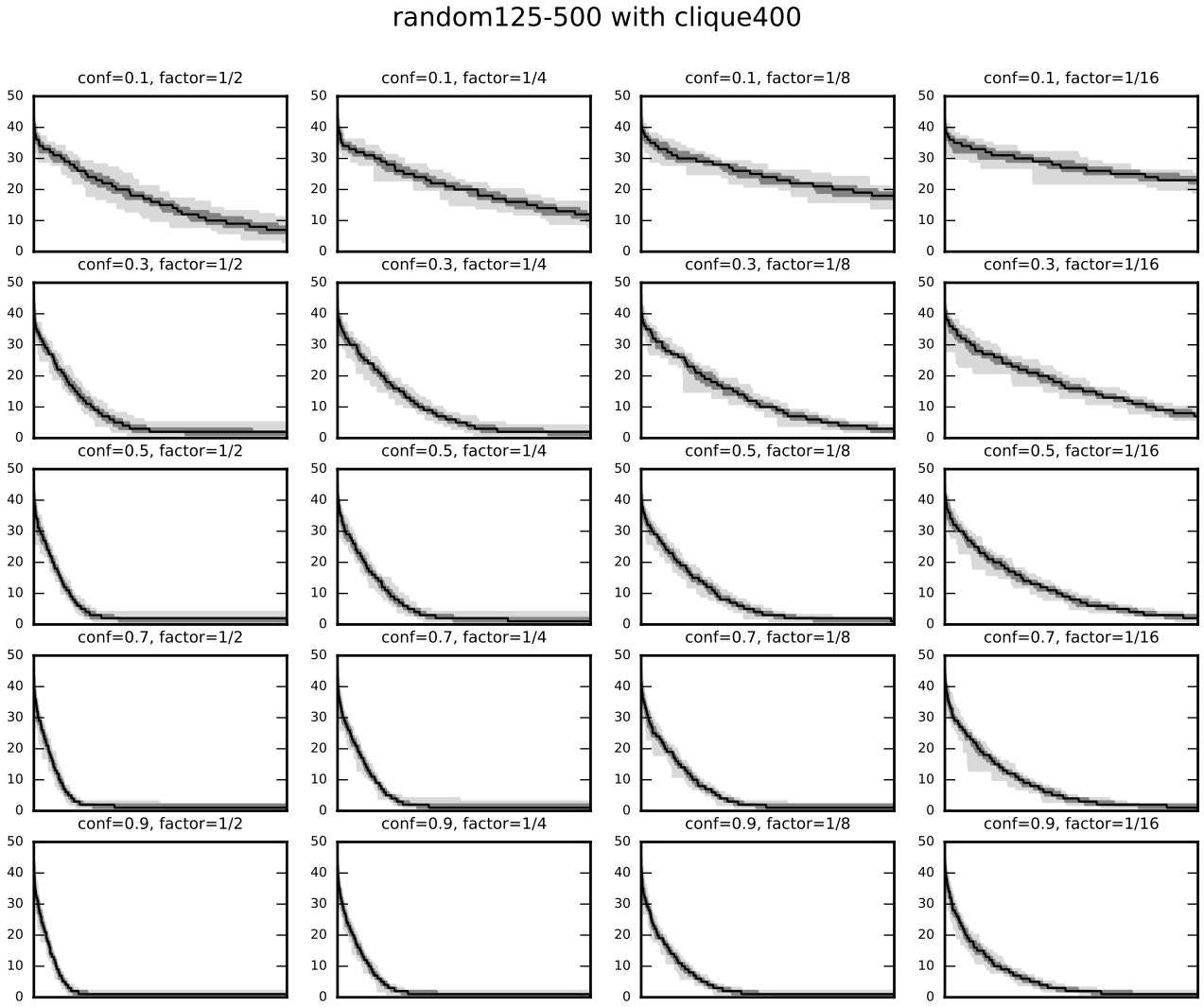


Figure 5.7: Convergence properties of the social interaction algorithm when solving the SAT problem and using clique on 400 vertices. We use a random SAT instance with 125 variables and 500 clauses. We can see that the bigger the self confidence is the faster the convergence is, and the bigger the meeting factor is the faster the convergence is.

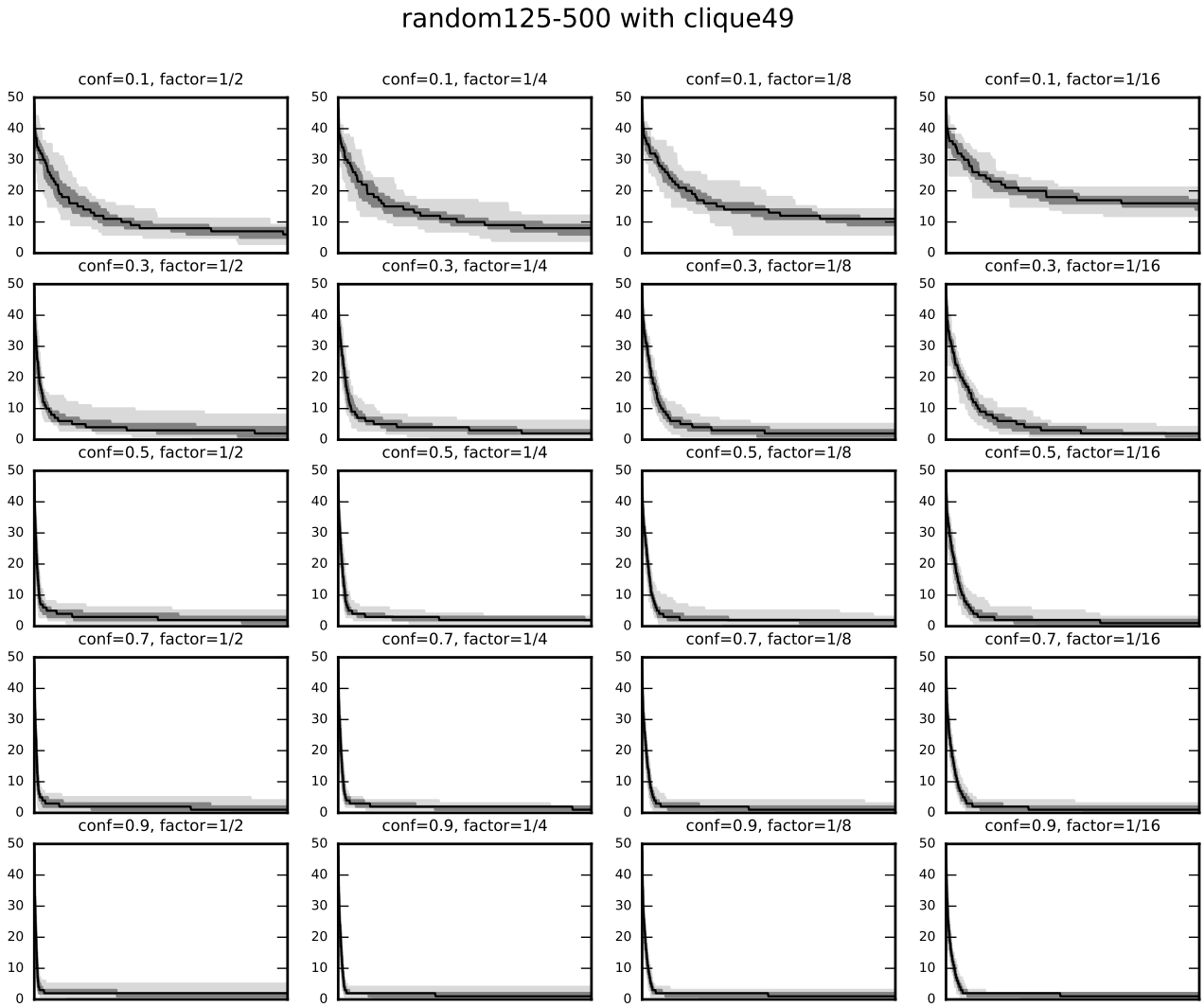


Figure 5.8: Convergence properties of the social interaction algorithm when solving the SAT problem and using clique on 49 vertices. We use a random SAT instance with 125 variables and 500 clauses. We can see that the bigger the self confidence is the faster the convergence is.

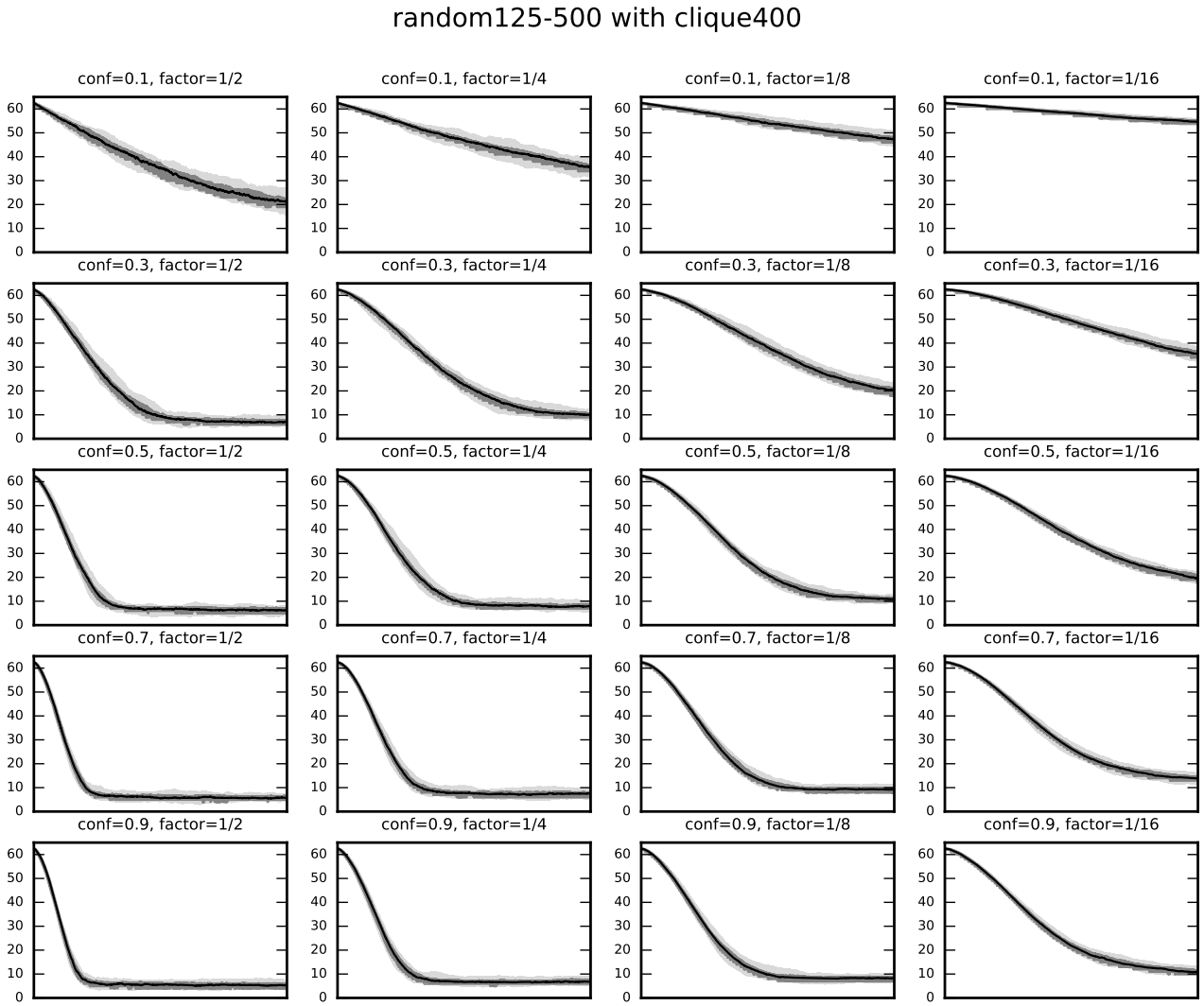


Figure 5.9: Genes variability of the social interaction algorithm when solving the SAT problem and using clique on 400 vertices. We use a random SAT instance with 125 variables and 500 clauses. The result is not surprising and it is similar to the result of the one-max problem. The only difference is that here it does not converge to zero but stays at a value from 5 to 10. It is probably caused by the fact that this SAT problem has more local optima.

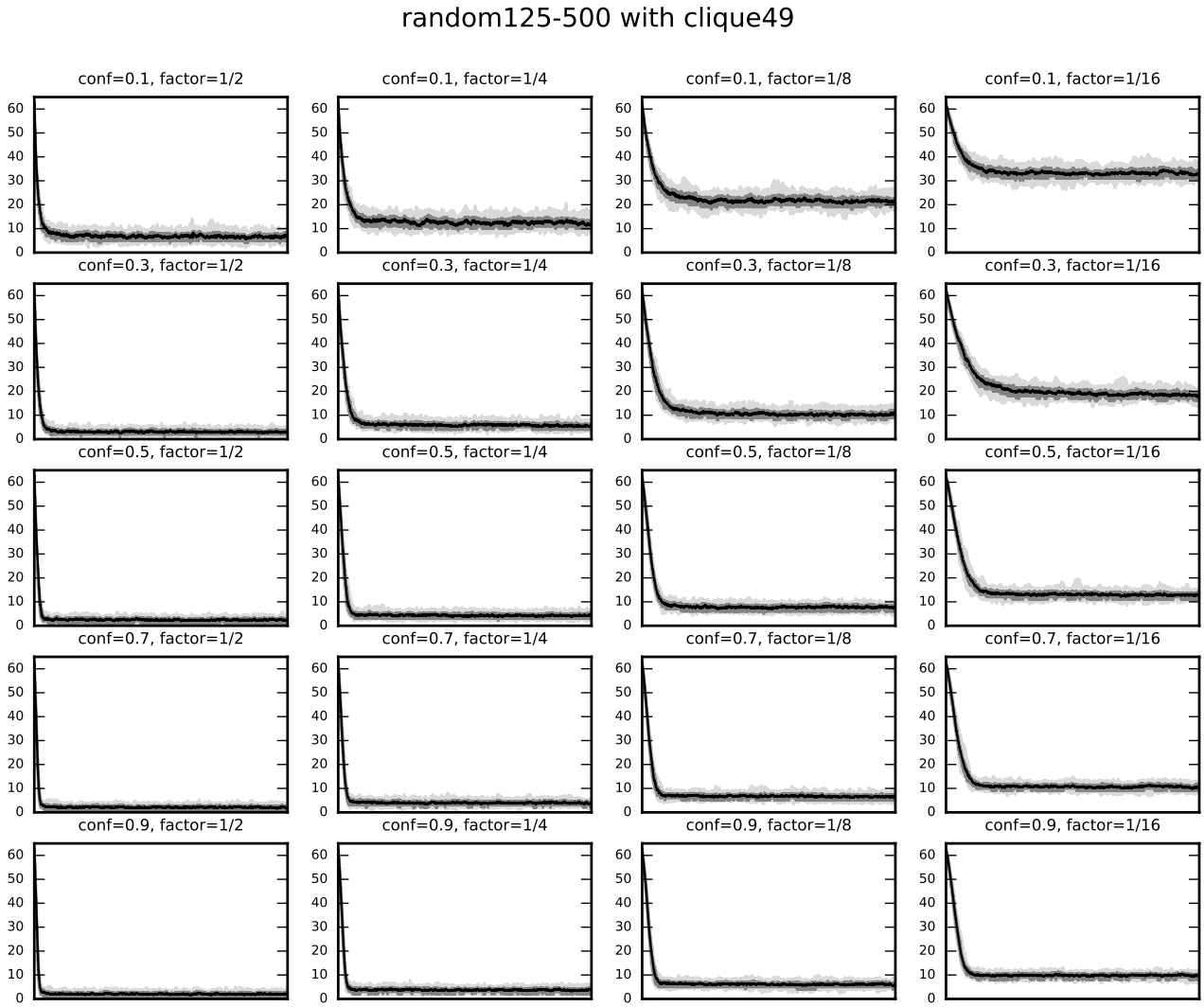


Figure 5.10: Genes variability of the social interaction algorithm when solving the SAT problem and using clique on 49 vertices. We use a random SAT instance with 125 variables and 500. We can conclude that the genes variability stays higher when a meeting factor is lower. And it holds even after the optimum solution is approaching. It is very advantageous property for solving complicated SAT problem instances.

	Clique		Circle		Double-circle		Ladder	
Fitness	median	best	median	best	median	best	median	best
32	1801	501	2201	1101	1701	401	1901	701
16	8201	6701	9201	5101	8001	5801	7701	3701
8	12001	10101	18300	12400	14400	9801	15101	11401
4	15301	12701	31601	25201	23101	17500	24001	15201
2	18900	16101	50600	31200	38501	23701	39501	21301
1	218700	17501	96400	45101	76200	39000	70300	25501
0	X	296900	X	99700	X	84200	X	68700

	Grid		Random		Random-clusters		Star	
Fitness	median	best	median	best	median	best	median	best
32	1901	500	1501	401	2001	701	901	100
16	8001	6401	7701	6001	7301	5201	4800	3501
8	14801	10801	12401	10701	13001	8801	7900	5101
4	20301	16701	16801	13101	25400	16501	9901	7001
2	24101	20801	23901	16601	53401	23101	15300	8801
1	29600	22401	268200	19900	122600	36000	450400	9300
0	X	39101	X	492200	X	138600	X	X

Table 5.9: Overview of the social interaction algorithm performance using various graphs with the self confidence $p_s = 0.9$, and the meeting factor $p_f = \frac{1}{2}$ on the random SAT instance with 125 variables and 500 clauses. We can see that a grid has the best performance followed by circular graphs and random-clusters.

	Clique		Circle		Double-circle		Ladder	
Fitness	median	best	median	best	median	best	median	best
32	3000	1201	3601	1701	3001	2101	2700	201
16	14801	8301	14800	12101	13301	9101	13001	9200
8	26200	21901	30800	20501	26100	19201	25101	16600
4	33900	29200	50300	31700	39000	24200	39800	24500
2	43100	36700	78000	47000	56000	31100	57300	24500
1	236400	38000	110300	61700	81201	50000	80500	57100
0	X	109200	X	73700	X	171100	X	113100

	Grid		Random		Random-clusters		Star	
Fitness	median	best	median	best	median	best	median	best
32	3101	1100	2501	1000	2801	901	401	101
16	13701	9401	12101	8101	13200	10401	5701	3901
8	25201	22000	23200	15101	23500	16801	13501	7801
4	36101	32301	31400	26500	33100	28101	18901	13100
2	48301	36101	41001	29701	50600	38800	24000	19101
1	57501	47400	49100	36600	72101	43000	49300	22701
0	X	66901	X	117400	X	118800	X	X

Table 5.10: Overview of the social interaction algorithm performance using various graphs with the self confidence $p_s = 0.9$, and the meeting factor $p_f = \frac{1}{8}$ on random SAT instance with 125 variables and 500 clauses. We can see that (surprisingly) a random graph and a grid have the best results followed by random clusters and circular graphs.

	Clique		Circle		Grid		Random-clusters	
Fitness	median	best	median	best	median	best	median	best
32	12401	7701	5801	3801	7001	3300	6201	3201
16	23001	17801	26301	18400	26601	19701	21401	13301
8	74100	24101	119301	74600	53401	40201	153201	41101
4	X	213500	X	156600	X	49701	X	180000
2	X	X	X	372401	X	X	X	395400
1	X	X	X	399000	X	X	X	X
0	X	X	X	399000	X	X	X	X

	Clique		Circle		Grid		Random-clusters	
Fitness	median	best	median	best	median	best	median	best
32	12601	5601	8600	4001	9301	4201	10001	4301
16	57201	45901	41901	27600	50700	37501	38701	31101
8	90901	64401	126900	81400	106900	66101	97600	61400
4	X	86000	337900	169200	228000	121901	X	151701
2	X	305600	X	293301	X	171301	X	488600
1	X	X	X	451901	X	373900	X	488600
0	X	X	X	X	X	X	X	X

Table 5.11: Performance of various graphs on SAT formula with 118 variables and 548 clauses that is based on factorization. The self confidence $p_s = 0.9$. In the top table there are results for meeting factor $p_f = \frac{1}{2}$, and in the bottom table there are results for meeting factor $p_f = \frac{1}{8}$. We can see that an algorithm with a lower meeting factor is able to converge further.

Algorithm	social interaction algorithm, genetic algorithm and neighbourhood model
Runs	25
Iterations	500,000
Problem instance	SAT problem with instances random125-500 and factor118-548
Fitness function	SAT fitness function
Population	49, 100, 400, 900
Graph type	clique, circle, ladder, grid, random-clusters
Self confidence	0.9
Meeting factor	$\frac{1}{2}, \frac{1}{8}$
Variability	local for all graphs except random and random-clusters for which we used global
Running time	1 hour, 58 minutes and 41 seconds

Table 5.12: Description of parameters of sat problem model comparison experiment.

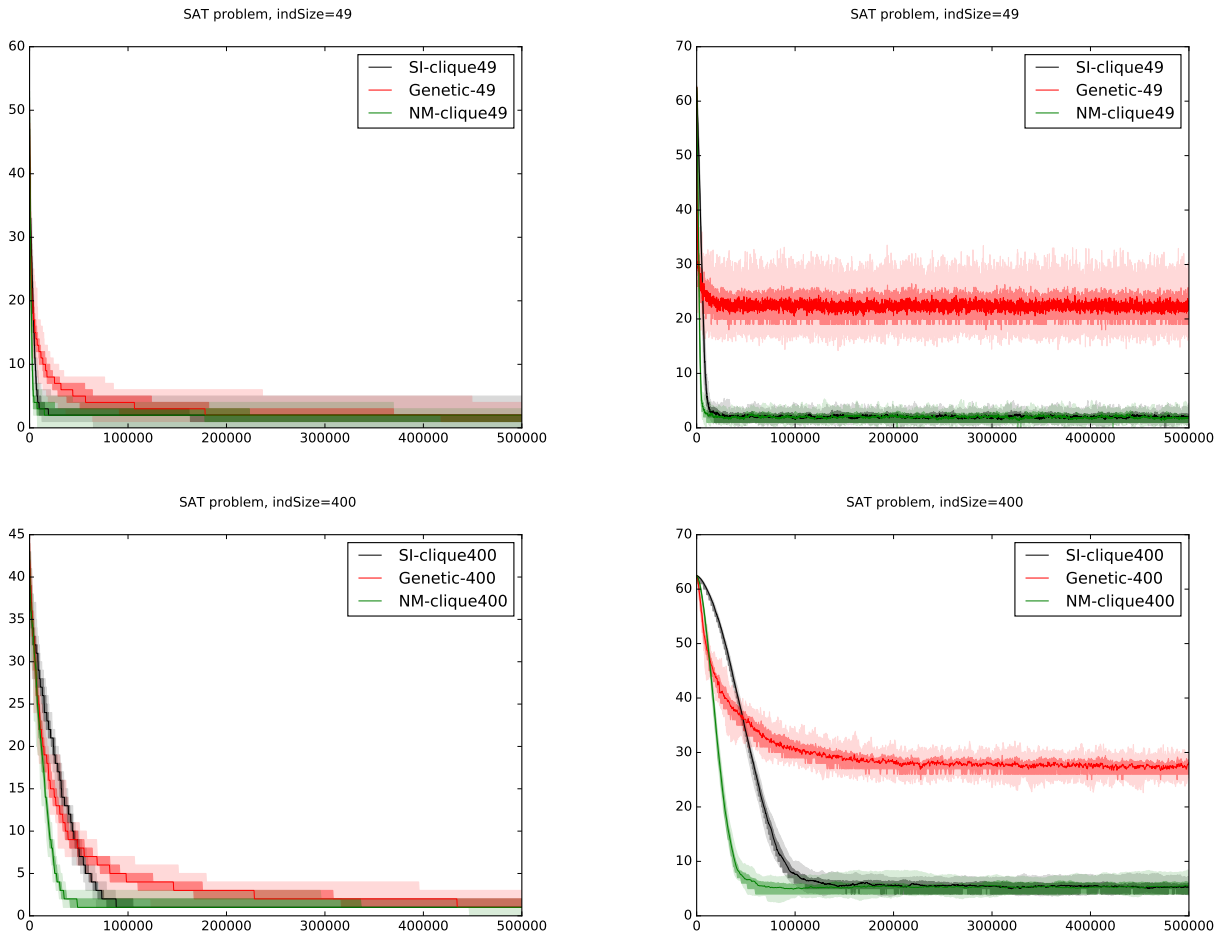


Figure 5.11: Model comparison when solving a random SAT with 125 variables and 500 clauses, and using clique of size 49 and 400. We can see convergence properties in the left graph and the variability convergence in the right graph. There is no big difference visible except that the genetic algorithm keeps higher variability at the end. This result does not give us any significant difference. See figure 5.12
to check the results when solving the SAT formula based on factorization, and more specialized parameter settings.

Now we use more specific parameter settings that is obligatory for each model, and compare the performance on SAT formula with 118 variables and 548 clauses that is based on factorization, and with population size $n = 100$. This SAT problem instance is more though so there is a better chance to differ those algorithms. For the neighbourhood model we use a grid, and for the social interaction algorithm we use a grid and random clusters. The result in figure 5.12 shows that the social interaction algorithm is more effective to solve tough SAT problems than both the genetic algorithm and the neighbourhood model.

Let us note that we test those models on more SAT problem instances, and for more iterations. For all non-trivial SAT problem instances the social interaction algorithm is significantly more successful than other models. We present just this one particular result as a demonstration.

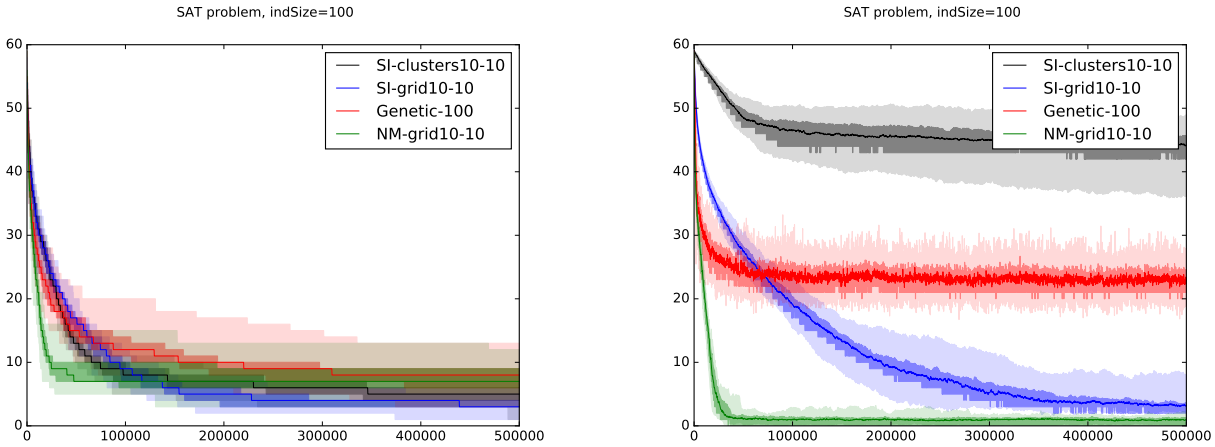


Figure 5.12: Model comparison when solving SAT formula with 125 variables and 500 clauses based on factorization, using population of 100 individuals. We compare the social interaction algorithm with random clusters, the social interaction algorithm with a grid, the neighbourhood model with a grid and the genetic algorithm. There are convergence properties in the left graph and variability convergence in the right graph. We can see that the best convergence properties has the social interaction algorithm using a grid, following by the social interaction algorithm using random clusters. The big difference between variabilities of both social interaction algorithms is there because we use the local variability in case of grid and the global variability in case of random clusters.

5.5 Clusters Types and Sizes on SAT problem

5.5.1 Motivation and Goals

As we introduced in section 4.3.1 we would like to design a graph which combines advantages of the clique and the circle. Our goal is to compare the performance of different types of cluster graphs, and different ratio of cluster sizes to the number of clusters for given population sizes. Then, we want to compare the performance to other graph types.

We are interested only in final results of every run, no matter how the convergence itself looks like. By the experiments 5.3 and 5.4 we know that the convergence is slower but it is able to end up close to the optimal value. For the purpose of this experiment, only the final best value of every run matters.

5.5.2 Experiment Description

We run the experiment for all combinations of parameters described in table 5.13. We use the following four SAT problem instances

- factor118-548: the SAT formula with 118 variables and 548 clauses that is based on factorization problem
- subsetsum194-837: the SAT formula with 194 variables and 837 clauses that is based on subset sum problem instance

Algorithm	social interaction algorithm
Runs	25
Iterations	2,500,000
Problem instance	SAT problem with instances factor118-548, subsetsum194-837, random256-1000, random512-2000
Fitness function	SAT fitness function
Population	105, 210
Graph type	cluster-circle, cluster-clique, cluster-doublecircle, random-clusters, weak-cluster-circle
Cluster-sizes	3, 7, 15, 21
Self confidence	0.9
Meeting factor	$[\frac{1}{16}, \frac{1}{4}]$[2] then there is picked a random integer b from interval $[\frac{1}{16}100, \frac{1}{4}100]$ and then b random bits takes a part in a discussion of individuals.
Variability	none
Running time	unknown[3]

Table 5.13: Description of parameters of clusters types and sizes experiment.

- random256-1000: randomly generated satisfiable SAT formula with 256 variables and 1000 clauses
- random512-2000: randomly generated satisfiable SAT formula with 512 variables and 2000 clauses

5.5.3 Results

In table 5.14 we can see the results for cluster graphs, and in table 5.15 we can see the results for other graph types. We provide an objective value of the best solution, median solution from 25 runs, and the variance of all 25 runs. We consider a solution A to be better than a solution B if its median is higher, the best solution is higher, and at the same time the variance is lower. By other words, we want a model that is able to consistently reach good solutions, and occasionally reach the optimal one.

Based on table 5.14 we can say that cluster-clique and weak-cluster-circle are in average less effective than other cluster types. For cluster-circle and weak-cluster-circle the best cluster size is 3 which then forms nearly the circle. For cluster-doublecircle and random-clusters the best cluster size seems to be 7 and sometimes 15 is good as well. The best result for all problem instances is reached by random-clusters, and in most cases with 30 clusters of size 7 which reaches very good results the most consistently.

When we look at results for other graph types in table 5.15, we can see that circular graphs like circle and double-circle have the best performance, and that

[2]This experiment was executed in older version of an algorithm. The only relevant difference is that the meeting factor was not given by probability p_f but by range $[p_{f_a}, p_{f_b}]$ that defines possible ratios of amount of changed bit during one meeting. For example when individual has size 100 and we have meeting factor given by the range $[\frac{1}{16}, \frac{1}{4}]$

[3]In this version of an algorithm we did not measure the running time.

the results are comparable but slightly better than results for cluster graphs. In next experiments we test other algorithm features that could help especially to cluster graphs.

5.6 Tie-Strength Application

5.6.1 Motivations and Goals

The goal is to test whether an application of the tie-strength in the social interaction algorithm in cluster graphs is advantageous or not. We want to determine the tie-strength influence for random-cluster graphs of various sizes and densities.

5.6.2 Experiment Description

We run the experiment with all combinations of parameters described in table 5.16. We use the same SAT problem instances as described in experiment 5.5. In addition, we test the performance for random clusters of various densities. Normally, when we generate random clusters, every node u with probability 0.5 choose another node v at random, and if u and v are from different clusters and are not already connected, we add an edge (u, v). We increase the density when we repeat this process several times for every node. Check the pseudo code in figure 4.5 in section 4.3.1 for a better imagination.

We test all densities for two cases. For the case when the tie-strength is not applied (standard version), and for the case when the tie-strength is applied. We observe an influence of two factors in results: the density and the tie-strength.

5.6.3 Results

The results are summarized in table 5.17. As results we provide the final result of median, the best of 25 runs, and the variance of 25 runs. We consider a solution A to be better than a solution B if its median is closer to zero, the best solution is closer to zero, and at the same time the variance is lower. By other words, we want a model that is able to consistently reach good solutions, and occasionally to reach the optimal one.

Let us first look at no tie-strength application, and the tie-strength application separately in dependency of the density level. When we look at clusters of size 7, we can see that when we use no tie-strength then the best performance is reached by density level 1, and the performance is decreasing while the density level is increasing. When we apply the tie-strength then the performance is increasing while the density is increasing as well. On the other hand, for cluster size 15 when we use no tie-strength then the performance is increasing up to density level 3 and then decreasing, and when we apply the tie-strength then the performance is just increasing.

We think that the important factor is the ratio of strong and weak ties usage. We can easily see that when strong ties are used too heavily then the performance is bad, as we can see for example in case of a cluster size 15 with the tie-strength

	factor118-548			subsetsum194-837			random256-1000			random512-2000		
Cluster type	med	best	var	med	best	var	med	best	var	med	best	var
circle-35-3	545	547	0.6496	835	836	0.7136	**997**	**999**	**1.2544**	**1996**	**1999**	**1.6576**
circle-15-7	545	548	1.7344	835	837	0.7584	997	999	1.4784	1996	1999	2.1824
circle-7-15	545	547	1.4944	835	837	0.7456	996	999	2.0544	**1996**	**1999**	**1.6736**
circle-70-3	**546**	**548**	**1.36**	**835**	**837**	**0.48**	997	999	1.3376	1995	1998	1.4816
circle-5-21	544	546	1.8784	835	836	0.3776	996	997	0.6336	1995	1997	0.9856
circle-30-7	545	547	0.9984	835	836	0.5376	996	999	0.8096	1995	1998	1.5616
circle-14-15	545	547	0.8224	835	836	0.64	996	999	1.2704	1995	1997	0.9856
circle-10-21	544	547	1.4016	835	836	0.5536	996	998	0.8896	1995	1997	1.5744
clique-35-3	544	547	2.2784	834	836	0.6336	**996**	**999**	**2.72**	1995	1997	2.3136
clique-15-7	543	546	2.3136	833	835	1.3344	994	998	2.48	1993	1997	3.3056
clique-7-15	543	546	3.7856	833	836	0.6496	995	999	3.12	1993	1996	2.5824
clique-5-21	544	548	2.5536	834	836	1.3216	995	997	1.1936	**1995**	**1999**	**2.64**
clique-70-3	544	545	1.4656	834	836	0.6336	995	998	1.92	1994	1996	2.0096
clique-30-7	**545**	**547**	**2.1664**	**834**	**836**	**0.5344**	995	997	0.7776	1994	1996	1.0816
clique-14-15	544	547	3.8176	834	834	0.7296	995	998	1.5616	1992	1994	1.9136
clique-10-21	543	547	3.4656	834	836	0.96	995	999	1.7344	1992	1995	2.3456
doublecircle-35-3	545	547	0.96	835	836	0.9504	996	999	1.68	1995	1998	1.7024
doublecircle-15-7	545	547	1.2224	835	836	0.5344	**997**	**999**	**1.0624**	**1996**	**1998**	**1.84**
doublecircle-7-15	545	547	1.0816	**835**	**837**	**0.4736**	996	999	1.4784	**1995**	**2000**	**2.2816**
doublecircle-5-21	544	547	1.0816	834	836	0.96	996	998	0.96	1995	1997	1.2544
doublecircle-70-3	545	547	1.2544	835	836	0.48	997	999	1.2896	1993	1996	2.8
doublecircle-30-7	**545**	**547**	**0.72**	835	836	0.6656	997	998	0.5536	1995	1997	0.8064
doublecircle-14-15	544	547	1.2704	834	836	0.5216	996	999	1.1936	1995	1998	1.6704
doublecircle-10-21	545	548	1.2704	834	836	0.7296	996	999	1.0944	1994	1996	1.8624
random-35-3	545	548	1.4144	835	836	1.1104	996	999	1.8144	1995	1998	1.6576
random-15-7	545	547	1.12	835	837	0.88	997	999	1.44	**1997**	**1998**	**1.0976**
random-7-15	544	546	1.2736	834	836	1.0624	996	999	1.44	1995	1996	1.8496
random-5-21	544	547	1.6416	834	836	1.3536	996	999	1.4624	1994	1997	2.5344
random-70-3	546	547	0.9024	835	837	0.7456	997	998	0.8096	1996	1998	1.0784
random-30-7	**546**	**548**	**1.1456**	**836**	**837**	**0.4**	**998**	**999**	**1.5264**	1996	1998	1.1584
random-14-15	544	547	1.6	835	836	1.04	996	997	0.72	1994	1997	1.9904
random-10-21	544	547	1.6256	834	835	0.48	995	998	1.2576	1993	1996	1.5904
weak-circle-35-3	**546**	**548**	**1.1136**	**835**	**836**	**0.3904**	996	999	1.1776	**1995**	**1997**	**1.7024**
weak-circle-15-7	544	547	1.4464	833	835	1.2	995	998	1.3504	1993	1995	1.8784
weak-circle-7-15	544	546	1.6224	832	834	0.88	995	997	1.1424	1991	1994	3.6736
weak-circle-5-21	543	548	2.9024	833	835	2.0864	995	997	1.4464	1991	1996	2
weak-circle-70-3	545	547	0.8896	**835**	**836**	**0.3424**	**996**	**999**	**1.0016**	1994	1997	1.12
weak-circle-30-7	544	546	1.28	832	834	1.0624	994	998	1.1104	1989	1994	4.1696
weak-circle-14-15	544	546	1.5616	831	834	1.68	994	995	0.5344	1989	1991	1.4944
weak-circle-10-21	544	546	1.3664	832	836	2.2976	995	997	0.96	1989	1995	4.1024

Table 5.14: Results for cluster graphs. For every type of graph and every problem, we can see the median, the best run, and the variance of all runs. Bold values label the cluster-size with the best, or nearly best result for every graph type and a problem instance separately. Red values label the best, or nearly the best result for every problem instance.

	factor118-548			subsetsum194-837			random256-1000			random512-2000		
Cluster type	med	best	var	med	best	var	med	best	var	med	best	var
cicle100	**546**	**548**	**1.0016**	**836**	**837**	**0.3776**	997	999	1.1584	1995	1998	2.4736
circle225	**546**	**548**	**0.9184**	836	836	0.3104	997	999	1.1456	1994	1997	1.5424
doublecircle100	546	548	1.2704	**836**	**837**	**0.4064**	**997**	**999**	**0.8**	1996	1998	1.76
doublecircle225	**546**	**548**	**0.72**	**836**	**837**	**0.4096**	**997**	**999**	**0.6944**	1994	1998	1.84
grid10-10	544	547	1.76	834	836	0.3744	996	999	2.16	1994	1997	1.9616
grid15-15	**546**	**548**	**0.9056**	835	836	0.5216	996	998	1.0464	**1997**	**1999**	**1.4016**
clique100	542	546	2.4416	834	836	2.08	994	999	2.8096	1993	1997	2.08
clique225	543	545	2.6496	834	834	1.12	995	998	2.6976	1992	1995	2.9856

Table 5.15: Results for other graph types. Bold values label results that are at least nearly as good as the best results of cluster graphs.

Algorithm	social interaction algorithm
Runs	25
Iterations	2,500,000
Problem instance	SAT problem of instances factor118-548, subsetsum194-837, random256-1000, random512-2000
Fitness function	SAT fitness function
Population	210, 315, 420
Graph type	random-clusters of cluster sizes 7 and 15
Self confidence	0.9
Meeting factor	$\frac{1}{8}$
Variability	none
Running time	123 hours, 20 minutes and 36 seconds

Table 5.16: Description of parameters of tie-strength application experiment.

application. And when we use weak ties too heavily then the performance is not optimal either, as we can see for a cluster size 7 with no tie-strength.

When we compare the performance of using, and not using the tie-strength then we can see that for cluster size 15 not using the tie-strength is always better. It is because the tie-strength cause strong ties to be used too heavily. This factor might disappear when we would use higher density level. In case of cluster size 7, for density levels 1 and 2 not using a tie-strength is usually better, and for density levels 4 and 5 using the tie-strength is usually better (it is visible the best in case of 30-7, bigger graphs might not converge perfectly).

5.7 Unique Self Confidence

5.7.1 Motivations and Goals

The motivation is to find out whether the usage of unique self confidence could increase the performance of the social interaction algorithm on the SAT problem. Based on experiment 5.3 we can say that high self confidence is advantageous so we test uniform distribution of self confidence, and another where we prefer higher self confidences to be used.

5.7.2 Experiment Description

We run the experiment with all combinations of parameters described in table 5.18 where we use the same SAT problem instances as in experiment 5.5. We use three different settings of the self confidence. All of them are also described in section 4.5.1.

0. **Standard**: every individual has the same self confidence $p_s = 0.9$.
1. **Uniform**: every individual picks the self confidence p_s uniformly randomly from $[0.05, 0.95]$.
2. **Better of two**: every individual picks two values uniformly randomly from $[0.05, 0.95]$, and then the bigger value is set as his or her self confidence p_s.

		factor118-548			subsetsum194-837			random256-1000			random512-2000		
Type	Clusters	med	best	var	med	best	var	med	best	var	med	best	var
NO 1	30-7	2	0	1.03	1	0	0.21	3	1	1.12	5	1	2.15
YES 1	30-7	5	2	1.35	5	3	1.02	6	2	1.11	11	9	1.95
NO 1	45-7	2	1	0.84	1	1	0.33	3	1	1.19	6	4	0.88
YES 1	45-7	5	1	1.36	6	4	1.47	7	4	1.03	14	11	1.68
NO 1	60-7	2	1	0.76	2	1	0.46	3	2	0.65	7	5	1.72
YES 1	60-7	4	3	0.73	7	6	0.76	7	5	0.74	17	12	3.12
NO 2	30-7	2	1	0.69	2	0	0.60	3	2	1.01	4	2	0.76
YES 2	30-7	4	2	0.56	4	1	1.56	5	3	0.83	9	6	1.57
NO 2	45-7	2	0	0.88	2	1	0.39	3	2	0.67	3	2	1.00
YES 2	45-7	4	3	0.75	4	3	1.21	6	3	1.28	11	7	2.56
NO 2	60-7	1	1	0.39	2	1	0.35	3	1	0.74	4	2	1.52
YES 2	60-7	5	2	1.21	6	3	1.69	7	4	1.13	14	9	2.10
NO 3	30-7	3	1	1.44	3	2	0.27	4	3	0.89	6	3	1.44
YES 3	30-7	3	1	0.84	3	2	0.91	4	2	1.00	8	5	0.97
NO 3	45-7	3	0	1.60	3	1	0.38	4	3	0.44	5	3	1.40
YES 3	45-7	4	2	0.65	4	1	0.92	5	2	1.32	9	6	2.36
NO 3	60-7	2	0	0.93	3	1	0.33	4	2	0.83	6	2	2.42
YES 3	60-7	4	3	0.40	4	3	0.66	6	3	0.87	11	7	2.50
NO 4	30-7	3	1	1.74	3	1	0.79	5	2	1.55	7	4	1.85
YES 4	30-7	2	1	0.73	3	1	0.78	4	1	1.05	7	3	1.36
NO 4	45-7	3	1	1.62	3	2	0.28	4	2	1.28	7	5	1.92
YES 4	45-7	3	0	1.76	3	1	0.47	4	2	1.03	8	4	1.52
NO 4	60-7	3	1	1.19	3	2	0.16	4	1	1.04	6	2	3.36
YES 4	60-7	3	0	1.02	4	1	1.11	5	4	0.58	10	6	2.81
NO 5	30-7	4	1	2.40	3	2	0.45	5	1	1.74	7	4	2.63
YES 5	30-7	3	0	0.81	2	1	0.28	4	1	1.29	5	3	1.00
NO 5	45-7	4	2	1.41	3	2	0.28	5	3	0.92	7	4	3.08
YES 5	45-7	3	1	1.01	2	1	0.69	4	2	1.23	7	5	1.08
NO 5	60-7	4	1	2.36	3	1	0.23	4	2	1.97	7	4	2.47
YES 5	60-7	3	0	1.24	3	1	0.38	5	2	0.97	8	6	1.99

		factor118-548			subsetsum194-837			random256-1000			random512-2000		
Type	Clusters	med	best	var	med	best	var	med	best	var	med	best	var
NO 1	14-15	3	1	1.27	3	1	0.50	4	2	1.44	7	4	1.96
YES 1	14-15	4	2	1.04	5	1	1.47	6	3	1.07	11	9	1.91
NO 1	21-15	4	0	1.60	3	2	0.52	4	2	1.78	8	4	2.25
YES 1	21-15	4	2	1.18	6	3	1.54	6	4	1.19	13	10	2.52
NO 1	28-15	4	2	0.67	3	2	0.50	5	3	0.82	8	5	1.69
YES 1	28-15	5	3	0.86	6	3	1.80	7	5	0.76	15	11	1.44
NO 2	14-15	3	0	1.40	2	1	0.48	3	2	0.84	5	3	0.86
YES 2	14-15	4	1	1.44	5	3	1.71	6	5	0.55	10	7	2.96
NO 2	21-15	3	0	1.33	2	1	0.51	4	2	0.86	6	4	0.80
YES 2	21-15	4	3	0.99	6	3	1.29	6	4	0.98	12	9	2.09
NO 2	28-15	3	1	1.28	2	1	0.43	4	2	0.92	6	4	0.75
YES 2	28-15	5	2	1.77	6	3	1.12	7	4	1.22	14	10	2.56
NO 3	14-15	2	1	1.08	2	1	0.75	3	2	0.74	4	2	1.01
YES 3	14-15	4	2	1.16	4	1	1.29	5	3	1.35	10	7	1.50
NO 3	21-15	2	1	0.84	1	1	0.49	3	1	0.68	4	3	0.45
YES 3	21-15	4	2	1.34	6	3	1.04	6	5	0.70	12	7	2.23
NO 3	28-15	2	0	1.32	2	1	0.36	3	2	0.52	5	3	1.60
YES 3	28-15	4	2	1.01	5	3	1.32	6	2	1.44	13	12	0.72
NO 4	14-15	2	0	1.24	3	1	0.63	4	1	1.22	4	2	1.23
YES 4	14-15	4	1	1.27	4	2	1.44	6	4	0.74	8	5	3.30
NO 4	21-15	2	0	1.17	3	1	0.48	3	1	1.06	4	2	0.91
YES 4	21-15	4	1	1.11	5	2	1.48	6	4	1.14	11	8	1.99
NO 4	28-15	2	1	0.83	2	1	0.63	3	2	0.47	4	2	0.91
YES 4	28-15	4	2	1.79	5	3	1.07	6	3	1.04	11	8	2.39
NO 5	14-15	3	1	1.15	3	1	0.36	4	3	0.49	6	3	1.67
YES 5	14-15	4	2	0.77	4	2	1.15	5	1	1.24	8	6	2.37
NO 5	21-15	2	1	1.23	3	1	0.34	4	1	0.91	6	2	2.25
YES 5	21-15	4	2	1.70	4	2	1.44	5	4	0.62	10	8	1.36
NO 5	28-15	2	0	1.03	3	1	0.40	4	2	0.71	6	3	1.29
YES 5	28-15	4	3	0.51	4	0	1.88	6	3	1.67	11	9	1.28

Table 5.17: Results of the tie-strength experiment for cluster sizes 7 and 15. In type column "NO" indicate that no tie-strength is applied and "YES" that the tie-strength is applied. The number determine the density level that is used.

Algorithm	social interaction algorithm
Runs	25
Iterations	2,500,000
Problem instance	SAT problem of instances factor118-548, subsetsum194-837, random256-1000, random512-2000
Fitness function	SAT fitness function
Population	105, 210
Graph type	clique, circle, ladder, grid, random-clusters of size 7
Self confidence	$[0.05, 0.95]$
Meeting factor	$\frac{1}{8}$
Variability	none
Running time	55 hours, 34 minutes and 51 seconds

Table 5.18: Description of parameters of unique self confidence experiment.

5.7.3 Results

Check table 5.19 with experiment results. We can see that the uniform random distribution has consistently find worse solutions than when the same self confidence is used. When higher self confidence values are preferred then results are better but still slightly worse than in case of the same self confidence.

Conclusion of this experiment is that uniqueness of self confidence (used in this way) probably does not have any positive significant influence on the performance of the social interaction algorithm.

5.8 Personal Bit Fitness Performance

5.8.1 Motivation and Goals

An opportunity to use a personalized fitness function is an important feature of the social interaction algorithm. The goal of this experiment is to examine the behaviour of personal bit fitness function for its different parameter settings.

5.8.2 Experiment Description

We run the algorithm with all combinations of parameters described in table 5.20. We use the same SAT problem instances as in experiment 5.5.

5.8.3 Results

Let us first briefly remind that the parameters setting (k, l, w) means that an individual has k his or her own random preferences, and takes l random preferences from every of his or her neighbours. Then, he or she adds a bonus w to his or her fitness value for every satisfied preference. A preference is a pair (b, v) and it is satisfied if and only if a bit number b equals to the value v.

Let us first look at results for lower values of k, l, and w that we can check in table 5.21. We can see that in cases where $w = 3$ (red, gray and brown rows), the results are almost always worse compared to the standard SAT fitness function

		factor118-548			subsetsum194-837			random256-1000			random512-2000		
Type	Graph	med	best	var	med	best	var	med	best	var	med	best	var
0	clique105	5	1	5.53	3	2	1.72	5	1	1.50	7	3	4.92
1	clique105	7	2	4.95	5	3	1.92	6	2	3.73	10	7	3.52
2	clique105	6	3	1.99	5	3	1.29	6	3	2.24	8	5	2.92
0	clique210	6	2	2.57	3	1	0.95	5	4	1.44	7	5	2.72
1	clique210	7	3	3.45	5	3	2.17	6	2	2.61	9	5	5.09
2	clique210	6	2	2.41	3	2	0.73	6	2	2.93	8	5	3.77
0	circle105	1	0	0.65	1	1	0.20	3	1	0.89	3	1	1.53
1	circle105	3	1	1.13	2	1	1.11	4	1	2.15	5	3	1.24
2	circle105	2	0	1.30	2	1	0.52	3	1	1.40	4	3	0.87
0	circle210	2	0	0.92	1	0	0.31	3	1	0.72	5	3	0.68
1	circle210	4	1	1.43	3	1	0.72	4	2	0.72	7	4	2.76
2	circle210	3	1	0.80	3	1	0.81	4	2	1.19	5	3	1.21
0	ladder106	2	0	1.00	1	0	0.60	3	1	0.75	4	1	1.64
1	ladder106	3	0	1.68	3	1	0.73	4	2	1.39	4	1	1.79
2	ladder106	2	1	0.72	1	1	0.48	3	1	1.38	3	0	2.56
0	ladder210	2	0	0.64	1	1	0.23	2	1	1.05	5	2	2.41
1	ladder210	3	2	0.63	2	1	0.39	4	1	1.55	6	3	1.75
2	ladder210	3	1	0.60	2	0	0.91	3	1	1.57	4	2	1.72
0	grid10-10	4	1	2.09	3	1	0.47	4	2	1.19	6	4	1.16
1	grid10-10	4	1	3.11	4	2	2.14	5	3	2.08	8	5	3.82
2	grid10-10	4	2	3.21	4	2	1.20	5	3	1.83	7	3	4.24
0	grid14-14	2	0	1.11	1	0	0.71	4	1	0.81	3	1	1.37
1	grid14-14	3	1	2.41	3	2	0.54	5	2	2.59	6	3	3.15
2	grid14-14	4	0	2.07	3	1	0.71	4	1	2.55	5	3	2.00
0	random-clusters15-7	2	0	1.28	1	0	0.57	3	1	1.28	3	1	0.93
1	random-clusters15-7	5	1	3.48	4	2	1.72	6	2	3.22	8	3	7.03
2	random-clusters15-7	4	1	3.08	3	0	1.84	5	2	1.13	5	2	2.36
0	random-clusters30-7	2	1	1.01	1	1	0.25	3	1	1.23	5	2	1.06
1	random-clusters30-7	4	1	1.40	3	1	0.63	5	2	2.02	6	2	3.99
2	random-clusters30-7	2	1	1.21	2	0	0.94	4	2	0.92	5	3	1.34

Table 5.19: Results of unique self confidence experiment. White rows are results when the same self confidence is used, in blue rows the uniform distribution is used, and in red rows the higher self confidence is preferred.

Algorithm	social interaction algorithm
Runs	25
Iterations	2,500,000
Problem instance	SAT problem of instances factor118-548, subsetsum194-837, random256-1000, random512-2000
Fitness function	Personal bit SAT fitness function for parameters (k,l,w) = [(0,0,0); (1,1,1), (1,1,3), (3,1,1), (3,1,3), (6,3,1), (6,3,3), (20,0,1), (50,0,1)]
Population	210
Graph type	circle, ladder, grid, cluster-circle with cluster size 3, random-clusters of size 7
Self confidence	0.9
Meeting factor	$\frac{1}{8}$
Variability	none
Running time	105 hours, 16 minutes and 24 seconds

Table 5.20: Description of parameters of personal bit fitness experiment.

		factor118-548			subsetsum194-837			random256-1000			random512-2000		
Params.	Graph	med	best	var	med	best	var	med	best	var	med	best	var
k=0,l=0,w=0	circle210	2	1	0.59	1	1	0.22	3	1	0.96	4	2	0.96
k=1,l=1,w=1	circle210	2	0	0.66	1	0	0.46	3	1	0.76	4	1	1.09
k=1,l=1,w=3	circle210	2	1	0.95	2	1	0.25	3	1	1.09	4	2	1.44
k=3,l=1,w=1	circle210	2	1	0.68	1	1	0.23	2	1	0.97	4	2	1.65
k=3,l=1,w=3	circle210	2	0	0.67	2	1	0.44	3	1	1.08	5	2	1.80
k=6,l=3,w=1	circle210	2	0	1.00	1	0	0.33	3	1	0.69	5	2	1.53
k=6,l=3,w=3	circle210	3	2	0.36	2	1	0.77	4	2	0.55	5	2	2.16
k=0,l=0,w=0	ladder210	2	0	0.71	1	0	0.28	3	1	1.22	4	3	0.89
k=1,l=1,w=1	ladder210	2	1	0.64	2	1	0.31	3	1	1.08	5	3	1.17
k=1,l=1,w=3	ladder210	3	0	1.03	2	1	0.72	3	1	1.12	5	1	2.31
k=3,l=1,w=1	ladder210	1	0	0.94	1	1	0.33	3	1	0.99	5	1	1.56
k=3,l=1,w=3	ladder210	2	0	0.77	1	0	0.71	3	2	0.72	5	2	1.05
k=6,l=3,w=1	ladder210	3	0	1.34	2	1	0.30	3	1	0.92	4	1	1.85
k=6,l=3,w=3	ladder210	3	1	1.00	3	0	0.89	3	2	0.95	5	2	1.31
k=0,l=0,w=0	grid14-14	2	1	0.95	2	1	0.83	4	2	0.73	4	1	1.59
k=1,l=1,w=1	grid14-14	2	1	1.08	2	1	0.50	3	1	1.19	4	1	1.53
k=1,l=1,w=3	grid14-14	2	0	0.98	2	1	0.71	3	1	1.16	4	1	2.12
k=3,l=1,w=1	grid14-14	1	0	0.96	2	1	0.63	3	1	1.16	4	3	0.48
k=3,l=1,w=3	grid14-14	2	0	1.17	2	1	0.64	3	1	1.42	4	2	1.55
k=6,l=3,w=1	grid14-14	2	0	0.84	2	0	1.07	3	1	1.65	4	3	1.20
k=6,l=3,w=3	grid14-14	2	1	0.96	2	1	0.79	3	1	1.27	4	1	1.66
k=0,l=0,w=0	cl-circ70-3	2	1	0.91	1	0	0.34	3	1	0.83	4	2	1.53
k=1,l=1,w=1	cl-circ70-3	2	0	0.88	2	1	0.40	3	1	1.40	5	2	1.52
k=1,l=1,w=3	cl-circ70-3	3	1	1.01	2	1	0.48	3	1	0.87	5	2	1.71
k=3,l=1,w=1	cl-circ70-3	2	0	0.92	1	0	0.57	3	2	0.62	4	3	1.10
k=3,l=1,w=3	cl-circ70-3	3	2	0.24	2	0	0.58	3	1	0.71	5	1	2.07
k=6,l=3,w=1	cl-circ70-3	2	1	0.71	2	1	0.39	3	1	1.40	5	2	1.49
k=6,l=3,w=3	cl-circ70-3	3	2	1.05	3	1	0.49	3	1	1.18	5	4	0.85
k=0,l=0,w=0	rand-cl30-7	2	0	1.67	1	0	0.38	3	1	0.96	4	3	0.79
k=1,l=1,w=1	rand-cl30-7	3	0	1.06	2	1	0.47	3	1	1.19	5	2	1.62
k=1,l=1,w=3	rand-cl30-7	3	2	0.73	3	1	0.65	4	1	0.87	6	4	1.79
k=3,l=1,w=1	rand-cl30-7	2	0	1.10	1	0	0.48	3	2	0.46	4	2	1.39
k=3,l=1,w=3	rand-cl30-7	3	1	1.15	2	0	0.68	4	2	0.84	5	3	1.79
k=6,l=3,w=1	rand-cl30-7	3	1	1.35	2	0	0.71	3	1	1.30	5	3	1.60
k=6,l=3,w=3	rand-cl30-7	6	4	1.24	6	4	1.03	5	3	1.69	8	5	1.21

Table 5.21: First part of personal bit fitness experiment. The table contains only results for lower values of (k, l, w). For the parameters setting $(0, 0, 0)$ is the fitness function is equivalent to the standard SAT fitness function with no personal preferences, and we use it mainly for the comparison.

(white rows). It is caused probably by the weight $w = 3$ because $w = 3$ is probably big enough to get stuck in local optima more often. So let us focus on cases where $w \leq 1$. We cannot see any significant differences between results. All the results are quite good and when we compare two different parameter settings (white, blue, green, yellow rows), we are always in the situation that sometimes one setting is better and other times the other setting is better.

In table 5.22 we can see the results where $k \in \{20, 50\}, l = 0, w = 1$. Unfortunately, again we cannot conclude anything because the results are just too similar. But still, we can observe at least another result that we were not looking for in this experiment. It is that random-clusters graph type is the most successful of all when we compare results independently on bit fitness parameters (color independently).

		factor118-548			subsetsum194-837			random256-1000			random512-2000		
Params.	Graph	med	best	var	med	best	var	med	best	var	med	best	var
k=0,l=0,w=0	circle210	2	1	0.59	1	1	0.22	3	1	0.96	4	2	0.96
k=20,l=0,w=1	circle210	2	1	0.51	1	0	0.31	3	1	0.96	4	2	1.60
k=50,l=0,w=1	circle210	2	1	0.67	2	1	0.50	3	1	0.64	5	2	1.19
k=0,l=0,w=0	ladder210	2	0	0.71	1	0	0.28	3	1	1.22	4	3	0.89
k=20,l=0,w=1	ladder210	2	0	0.76	2	1	0.32	3	1	1.00	5	2	1.75
k=50,l=0,w=1	ladder210	2	1	0.86	1	1	0.48	3	1	1.06	5	2	1.88
k=0,l=0,w=0	grid14-14	2	1	0.95	2	1	0.83	4	2	0.73	4	1	1.59
k=20,l=0,w=1	grid14-14	2	1	1.25	2	1	0.55	3	1	0.88	4	2	0.88
k=50,l=0,w=1	grid14-14	1	0	0.86	2	1	0.61	3	1	0.90	4	2	1.25
k=0,l=0,w=0	cl-circ70-3	2	1	0.91	1	0	0.34	3	1	0.83	4	2	1.53
k=20,l=0,w=1	cl-circ70-3	2	0	1.23	1	0	0.32	3	2	0.87	5	2	1.68
k=50,l=0,w=1	cl-circ70-3	2	1	0.64	2	1	0.46	3	2	0.64	5	3	1.20
k=0,l=0,w=0	rand-cl30-7	2	0	1.67	1	0	0.38	3	1	0.96	4	3	0.79
k=20,l=0,w=1	rand-cl30-7	2	0	0.94	2	0	0.49	3	1	0.91	4	2	1.18
k=50,l=0,w=1	rand-cl30-7	3	1	0.80	1	1	0.32	3	1	1.32	5	2	1.18

Table 5.22: Second part of personal bit fitness experiment. The table contains only results for high values of k, $l = 0$ and $w = 1$. For the parameters setting $(0, 0, 0)$ the fitness function is equivalent to the standard SAT fitness function with no personal preferences, and we use it mainly for the comparison.

Chapter 6

Conclusion

6.1 Theoretical Results Summary

In this thesis we have introduced the social interaction algorithm together with all its motivations and background. We have successfully joined knowledge of optimization search, especially nature-inspired optimization algorithms, together with social networks understanding, and developed the new algorithm that is non-trivially inspired by both areas. We consider the theoretical basis to be the main and the most important result of this thesis.

The main components of the algorithm are the *graph structure* that allows interaction only to certain pairs of individuals and the *meeting operator* that defines how the interactions look like and how individuals exchange information. We have also analyzed its complexity, and concluded that the most significant factor, which we can affect, is the total number of iterations that the algorithm needs to reach a given precision.[1] We have discussed its exploration and exploitation components as well as the influence of single parameters within them. Then, we suggested how and why it differs from other existing models. The main differences are the *graph infrastructure* that crucially influences the behaviour, *no explicit selection* that opens possibilities for individuality expression, and *an opportunity of individuality* itself that allows single individuals to be specialized on some local subspaces.

We have shown that the social interaction algorithm can mimic the Hill climbing method by theorem 1 and we have proven the theorem 4 which says that the social interaction algorithm can run effectively in parallel using $\mathcal{O}(\sqrt{\frac{m}{\Delta}})$ threads where m is the number of edges in a graph and Δ is the maximum degree in a graph. Especially regular graphs can use effectively $\mathcal{O}(\sqrt{n})$ threads in parallel where n is the number of vertices as well as the population size.

6.2 Experimental Results Summary

We did basic parameters tuning for the one-max problem and for the SAT problem. We examined that it is good to keep the self confidence p_s high (i.e.

[1] We have shown that in reasonable cases the number of iterations overwhelms other factors of the time complexity with the exception of the fitness function complexity. But the fitness function we have given together with the problem, so we cannot affect it.

$p_s = 0.9$) and that the self confidence is a crucially important factor for good convergence properties. We also found out that when the meeting factor p_f is increasing, the convergence speed is decreasing so is the speed of the genes variability convergence. And even when the convergence was slower at the beginning, the algorithm was able to converge further in the finishing phase and obtain better results at the end. A further convergence is supported by the slow variability convergence, and the further convergence means better results, so we usually set the meeting factor to be lower (i. e. $p_f = \frac{1}{8}$).

An important parameter is the choice of a graph. We concluded that, especially for more complicated and non-trivial problems, sparse graphs with circular structure and clustered graphs lead to better performance. So we paid more attention to those two classes. Circular graphs like circle, grid, or ladder have consistently good results as well as clustered graphs. Clustered graphs are more complex and more promising for the usage of other algorithm features, like for example individuality, so we decided to explore them more.

The conclusion about clustered graphs is that the best type of clustered graphs is random clusters which are the most similar to the real social network structure from all the types we used. It turned out that higher number of smaller clusters is advantageous, in our case 30 clusters of size 7 for middle-sized SAT problems. We continued by tie-strength application in random clustered graph. The conclusion is that the tie-strength is advantageous when it causes a good ratio between strong and weak ties. It means that when we have low number of weak ties, then we should not apply the tie-strength, and when we have high number of weak ties, then we should use some tie-strength to balance the ratio. Again, such property corresponds with knowledge from real social networks, they have huge number of weak ties but strong ties are used more often.

Unfortunately, we have not shown any evidence of the importance of individuality. We tried to use unique self confidence and personalised fitness functions but the results were not significantly better or worse. We think that it is partially because of the fact that our initial results, given by the basic version of the social interaction algorithm, were too good to be beaten easily. We still believe that for other, possibly bigger problems, the individuality can play an important role.

6.3 Future work

Many possibilities for future research are open, especially in the area of individuality. We did all experiments on a general basis, and we did all the graph generation and settings of individuals randomly. It could be worth researching the issue of non-random graphs and individuals more concretely. For example, to design special kinds of individuals as we suggested in section 4.5.3, and to find a good way of cooperation for these types of individuals. Or to examine which individuals should be included in a cluster to reach the best results, or how the different types of individuals should be connected together to cooperate the best, et cetera.

It might be interesting to try to extend the social interaction algorithm to solve continuous optimization problems. At this point it is not clear how to do it. We believe that by a proper modification of the meeting operator, and maybe by another inspiration in human interactions, it is possible to develop a version

of the social interaction algorithm that works for continuous problems as well.

Another direction of the research is more detailed examination of clustered graphs. Several times we concluded that a clustered graph works the best when it fulfils the same property as social networks. So, when continuing the research, we could end up with an optimization algorithm, that performs the best when using a graph mimicking a real social network. This conclusion can be expressed by the following optimistic hypothesis.

Hypothesis 5. *There exists a social interaction-like algorithm and its setting, such that the algorithm shows the best results when using a graph mimicking a social network. When it does, its performance is significantly better than other algorithms of a similar type.*

Bibliography

[1] Xin-She Yang. *Nature-Inspired Optimization Algorithms*. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, 1st edition, 2014.

[2] Jennifer Golbeck. *Analyzing the Social Web*. Morgan Kaufmann, Elsevier, 1st edition, 2013.

[3] M. Gladwell. *The Tipping Point: How Little Things Can Make a Big Difference*. Little, Brown, 2006.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[5] Walter Rudin. *Principles of mathematical analysis*. McGraw-Hill Book Co., New York, third edition, 1976. International Series in Pure and Applied Mathematics.

[6] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003.

[7] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983.

[9] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statist. Sci.*, 8(1):10–15, 02 1993.

[10] Marco Dorigo, Gianni Di Caro, and Luca M. Gambardella. Ant algorithms for discrete optimization. *Artif. Life*, 5(2):137–172, April 1999.

[11] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (abc) algorithm. *J. of Global Optimization*, 39(3):459–471, November 2007.

[12] John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.

[13] Darrell Whitley and Timothy Starkweather. Genitor ii.: A distributed genetic algorithm. *J. Exp. Theor. Artif. Intell.*, 2(3):189–214, October 1990.

[14] Paul Bryant Grosso. *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model*. PhD thesis, Ann Arbor, MI, USA, 1985. AAI8520908.

[15] H. Kwasnicka and W. Kwasnicky. The importance of recombination in speciation and evolution of the higher taxa. 1987.

[16] Martina Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, PPSN I, pages 150–159, London, UK, UK, 1991. Springer-Verlag.

[17] J. David Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms*, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[18] Donald Macfarlane and Ian East. An investigation of several parallel genetic algorithms. In *Proceedings of the 12th Occam User Group Technical Meeting on Tools and Techniques for Transputer Applications*, OUG-12, pages 60–67, Amsterdam, The Netherlands, The Netherlands, 1990. IOS Press.

[19] J.D. Schaffer and L.J. Eshelman. On Crossover as an Evolutionary Viable Strategy. In R.K. Belew and L.B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 61–68. Morgan Kaufmann, 1991.

[20] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[21] Satu Elisa Schaeffer. Survey: Graph clustering. *Comput. Sci. Rev.*, 1(1):27–64, August 2007.

[22] Stanley Milgram. The small world problem. *Psychology Today*, 67(1):61–67, 1967.

[23] Three and a half degrees of separation, 2016.

[24] M.S. Granovetter. The Strength of Weak Ties. *The American Journal of Sociology*, 78(6):1360–1380, 1973.