# Configuring Genetic Algorithm to Solve the Inverse Heat Conduction Problem

**Sándor Szénási, Imre Felde**

Óbuda University, Bécsi út 96/b, H-1034 Budapest, Hungary
E-mail: szenasi.sandor@nik.uni-obuda.hu, felde.imre@nik.uni-obuda.hu

*Abstract: Accurate design of heat treatment operations requires the knowledge of the Heat Transfer Coefficients (HTC), which quantity can be determined by performing Inverse Heat Transfer Calculations. The novel approaches for the estimation of HTC are based on heuristic optimisation methods, but the usage of these techniques raises several questions. In the case of genetic algorithms, there are not any rules of thumb for selecting the appropriate population size, mutation rate, stopping condition, and similar. The most efficient way to fine-tune these parameters is to run thousands of experimental tests and evaluate the results. However, in the case of inverse heat conduction, this has not been a viable option because of the high computational demand of fitness calculation which leads to a runtime of dozens of years. This paper presents a solution to this problem using a novel data-parallel direct heat conduction problem solver method implemented on multiple graphics accelerators. The ~100× speed-up achieved by this parallel algorithm made it possible to finish the necessary experimental tests in 15 weeks (instead of 29 years). Data gathered during these experiments are directly useful in practice. Based on these, it is possible to make recommendations for optimal genetic algorithm configuration parameters.*

*Keywords: inverse heat conduction problem; genetic algorithm; parameter optimisation; population size; mutation rate; stopping condition; graphics accelerators; CUDA*

# 1 Introduction

It is a well-known fundamental experience that material properties are not constantly determined by their chemical composition, but they are influenced by their microstructure [1]. Heat treatment (heating up the object to a specific temperature and cooling it down under strict temperature control) is one of the most efficient methods to produce the desired microstructure of the material.

The proper design of the heat treatment process requires an accurate knowledge of the thermal boundary conditions, including the Heat Flux (HF) or the Heat Transfer Coefficient (HTC). HTC describes heat exchange between the surface of

an object and the surrounding medium. The determination of HTC faces a typical Inverse Heat Conduction Problem (IHCP). The IHCP methods are using the temperature signals recorded and estimated by simulations at given locations of the work-piece in order to predict HTC functions. Several IHCP approaches are based on optimization methods, where the goal function has to be minimised is given as the deviation of the measured and predicted temperature data [2], [3].

It is usual to solve this problem with some kinds of heuristic search algorithms, like Genetic Algorithms (GAs) [4]–[6] or Particle Swarm Optimisation (PSO) [4], [7]–[9]. In the case of GAs, every chromosome encodes one possible HTC function. Using the Direct Heat Conduction Problem (DHCP) calculations, it is feasible to generate the theoretical thermal history based on each HTC variants; and it is also possible to compare this theoretical history to the real measured temperature values. Our goal is to find the HTC function generating the cooling curve, which is the most similar to the measured one.

Beyond the generally known challenges of these heuristic methods (stability, unpredictable convergence, and others), the applications raise several technical issues:

- It is difficult to choose the appropriate variant for a practical task [10].
- Moreover, after selecting the suitable method, there are not any unequivocal rules to set the best working parameters (population size, mutation probability, and stopping condition) [11], [12].

The proposed methodology is based on an evaluation of virtual tests generated by a huge variety of different configurations (including population size, mutation probability/rate, and stopping condition). Due to the high computational requirements of IHCP solvers, parallel processing DHCP is suggested. A novel data-parallel DHCP solver has been developed and implemented as a GPU application. The recommendations for the appropriate genetic algorithm configurations are given by the results of the computational experiments.

The rest of this paper is structured as follows: Section 2 contains details of the problem formulation based on the GA approach, and Section 3 presents the issues raised by fitness calculation and a brief presentation about the already mentioned GPU based DHCP solver implementation. The final section focuses on the methodology followed by the experimental tests and the conclusions.

# 2 Methodology

## 2.1 Genetic Algorithms

GA is one of the most popular biologically inspired methods based on the language of natural genetics and biological evolution. As a heuristic search method, it uses a set of individuals (chromosomes) referred to as a population. Every chromosome represents a potential solution for the raised problem. Chromosomes encode the corresponding potential solution in their genes as numbers. After a random initialization, these genes change according to a previously fixed rule set when trying to find the optimal solution for the original problem.

A full GA search needs several iterations, where every iteration consists of the same main steps. Randomly selected pairs (given by the selection operator) of individuals are mated using a process of crossover (using the crossover operator). As a result of these steps, new individuals inherit genes from either parent. To help the algorithm map as large a part of the parameter space as possible, these individuals undergo a random mutation, in which some of their genes change by a random amount (specified by the mutation operator).

An essential part of the method is the fitness calculation. This assigns a comparable value to every chromosome based on their genes. This fitness value plays a major role in the selection phase (individuals with better fitness value usually have a higher probability of being selected). It is also common to use these fitness values to set up some form of stopping condition. For example, the GA stops when the increase of the best fitness value slows down.

## 2.2 Problem Formulation

### 2.2.1 Chromosome Representation

Each chromosome represents a possible solution for the problem. There are several methods for chromosome representation (how the information is encoded into the genes) based on the problem itself (input data characteristics – type, amount, structure, and similar.) and the chosen implementation.

In the case of 2D IHCP, the feasible solutions are the potential HTC functions. These are two-dimensional functions, where the actual value depends on the local coordinate and the time (Eq. 1).

$$HTC_{z,t} = ?\left|0 \le z \le L, 0 \le t \le t_{end}\right. \tag{1}$$

Where

- $z$ – local coordinate;
- $L$ – length of the work object;
- $t$ – time;
- $t_{end}$ – time period of the experiment.

Although it is a continuous function, the discrete DHCP process uses some simplifications:

- Local coordinate discretization: if the DHCP process uses an $n \times m$ sized matrix to simulate the heat movement inside the work-piece (see Section 3.1), only the positions $0$, $L/m$, $2L/m$, …, $L$ are used as local coordinates.
- Time discretization: the DHCP process starts the simulation at time point 0 and executes an elementary heat-transfer step for every $dt$ second. Therefore, it will need the HTC values at time coordinates 0, $dt$, $2dt$, …, $t_{end}$.

Theoretically, it is possible to encode all of these HTC values as separate floating point values in the genes, but this leads to an unmanageably large number of parameters. To significantly decrease the size of the parameter space, it is enough to store fewer parameters as control points and use bilinear interpolation to estimate the necessary values:

- For local coordinate discretization, we store the HTC values for only $Z$ number of local coordinates (where $Z < m$). These positions are fixed in advance, conveniently according to the positions of thermocouples built inside the real-world object.
- A dynamic resolution model is used to reduce the number of necessary time coordinates. This means that we use K control points (according to different time moments) in each location; and as a novel idea, the number of $K$ varies during the search. In the first phase, $K$ equals to 3 to find a rough estimation. When the genetic algorithm cannot fine-tune the best solution with this setting, it increases the value of $K$ to 5 and continues the search. One search contains 4 phases, using $K_1=3$, $K_2=5$, $K_3=9$, $K_4=17$ values. Consequently, every chromosome encodes Z×K control points; where every control point consists of two floats: time and HTC values (location values are fixed in advance).

### 2.2.2 Initialization

The first step of any heuristic algorithm is to initialize the elements (chromosomes, particles). As an essential concept, it is assumed that there is no prior information about the characteristics of the searched HTC function.

Therefore, the only possible way is to generate random numbers for the initial gene values based on the following rules:

- genes containing HTC values should have a random value between 0 and 7000 ($W/m^2/K$);
- genes containing time values should have a random value between 0 and 180 (sec).

To find the ideal number of chromosomes in a population is one of our goals. As a free parameter, we will reference this number as $|P|$. Section 4.2.1 presents the details of the tests related to this value.

### 2.2.3 Elitism, Selection and Crossover Operators

Because of the large search space, we would like to guarantee that the best individuals will survive. To ensure this, the elitism technique [13] is used to move the top 10% of the chromosomes (having the best fitness values) to the next generation without crossover or mutation. As a side-effect, the fitness values for the best chromosomes become monotonically increasing.

In the next step, the selection operator is responsible for randomly choosing individuals from the population as parents. Individuals with better fitness values will be selected with a greater probability than others (chromosomes involved in elitism are still able to be selected as parents). Our implementation uses fitness proportionate selection (also known as the roulette wheel selection), where the fitness level of chromosomes is used to associate a probability of selection (Eq. 2).

$$p_i = \frac{\left| f_i - f_{best} \right|}{\sum_{j=1}^{|P|} \left| f_j - f_{best} \right|} \tag{2}$$

Where

- $p_i$ = probability to select the $i$-th chromosome for crossover;
- $f_i$ = fitness value for the $i$-th chromosome;
- $f_{best}$ = the best fitness value of the population;
- $|P|$ = number of chromosomes.

When preparing the chromosomes to the next generation, it is necessary to select two different ones based on the given probabilities and to use the crossover operation on these to create one offspring for the next generation. The uniform crossover operation [14] is used on the selected parents, which means that each gene (in this particular case, a float value) of the offspring is randomly picked from either of the two parent genes from the same position (the probability of inheriting from either of the parents is 50-50%). This means that there is no linkage between genes, the inheritance of these is independent from the others.

We must repeat the presented selection and crossover steps until the next generation of $|P|$ instances has been created.

### 2.2.4 Mutation Operator

To ensure exploration, it is necessary to use some kinds of mutation operators. Accordingly, every newly created (by selection and cross-over) chromosome of the next generation goes through an additional random process which can modify its genes. The proposed algorithm uses gene level mutations.

This means that each gene of the new chromosome can be changed with a certain probability. It is hard to determine the appropriate probability rates and extent of the changes. Finding these is one the primary goals of this research.

In all experiments, we use three levels of mutations:

- Large mutation:
  - probability: $M_L$ $(0 \leq M_L \leq 1)$,
  - rate of change: random value between $-R_L \ldots +R_L$.
- Medium mutation:
  - probability: $M_M$ $(0 \leq M_M \leq 1)$,
  - rate of change: random value between $-R_M \ldots +R_M$.
- Small mutation:
  - probability: $M_S$ $(0 \leq M_S \leq 1)$,
  - rate of change: random value between $-R_S \ldots +R_S$.

Section 4.2.2 presents the details of the tests relating to these values.

### 2.2.5 Stopping Criteria

The stopping criteria give the answer to the question raised at the end of every iteration - whether it is worth continuing the search or not. Obviously, if one of the chromosomes reaches the theoretically best fitness value, then it contains the desired solution, and it is necessary to stop the search. However, due to the nature of heuristic algorithms, there is no guarantee that this is going to happen. Therefore, this criterion is insufficient.

Another approach is to wait until the GA cannot produce any progress with respect to the best fitness of the population. One of the characteristics of this technology is that in most cases, the best fitness value of the population converges to the desired fitness value more and more slowly. Because of the unique dynamic resolution model, it also requires investigation than when it is worth to switch to the next phase.

Setting up these limitations is hard, so, the implementation uses two free parameters to handle this: the search will start the next phase when the achieved improvement of the best fitness value during the last $STOP_{iter}$ number of iterations is less than $STOP_{rate}$.

Section 4.2.3 presents the details of the tests relating to these values.

# 3 GPU-based Fitness Calculation

## 3.1 Fitness Definition

The result of the fitness function depends on the difference between the generated thermal history (using the DHCP solver and the HTC value encoded into the chromosome) and the measured temperature signals. This section presents the details of this DHCP solver.

A two-dimensional axis-symmetrical model is considered to estimate the temperature distribution in a cylindrical work-piece. The mathematical formulation of the nonlinear transient heat conduction problem can be described as follows (Eq. 3):

$$\frac{\partial}{\partial r}\left(k\frac{\partial T}{\partial r}\right)+\frac{k}{r}\frac{\partial T}{\partial r}+\frac{\partial}{\partial z}\left(k\frac{\partial T}{\partial z}\right)+q_v=\rho C_p\frac{\partial T}{\partial t} \tag{3}$$

With the following initial and boundary conditions (Eq. 4-5):

$$T(r,z,0)=T_0 \tag{4}$$

$$k\frac{\partial T}{\partial z}\bigg|_{\substack{0\le z\le L\\ r=R}}=HTC(z,t)\left[T_q-T(r,z,t)\right] \tag{5}$$

Where

- $r$, $z$ – local coordinates;
- $t$ – time;
- $R$ – radius of the workpiece;
- $\rho$ – density of the object;
- $T_0$ – initial temperature of the workpiece;
- $T_q$ – temperature of the cooling medium;
- $T(r,z,t)$ – temperature of the workpiece at given location/time;

- $k(T)$ – thermal conductivity (varying with temperature);
- $C_p(T)$ – heat capacity (varying with temperature);
- $HTC(z,t)$ – heat conduction (varying with local coordinate and time).

The solution of (Eq. 3) is obtained by a weighted Schmidt explicit finite difference method.

The fitness value of the individual is determined as the deviation between the measured and generated thermal history. Based on these values and the results of the explicit finite difference method, the fitness value for a given HTC is (Eq. 6):

$$F = \sum_{k=1}^{N} \left( T_k^m - T_k^c \right)^2 = \min \tag{6}$$

Where

- $N$ –total number of measured temperatures (the number of points multiplied by the number of measurements at each point);
- $T_k^m$ – measured values;
- $T_k^c$ – calculated values.

Our goal is to find the best HTC with minimal fitness value.

## 3.2 GPU-based Implementation

By using the finite difference method, it is possible to calculate the heat transfer between two points for a given small time step. To be able to ensure accuracy, a sufficiently small-time interval is necessary ($dt \leq 0.01$ sec). According to real-world measurements, it is usually required to continue the simulation of the cooling process for an extended period ($t_{end} \geq 120$ sec). For this purpose, the algorithm has to run the calculations mentioned above in a loop to specify the heat movement between the finite items for each time steps.

As visible, one DHCP calculation needs a lot of iterations (iteration count $\geq 120 / 0.01 = 12000$). Using a traditional CPU-based sequential algorithm, it takes about 0.24 sec to calculate all the necessary $T_k^c$ values and to specify the fitness value for a given chromosome.

According to the GA operating rules, we have to calculate the fitness value for every chromosome in every iteration. To achieve our goals, we have to launch hundreds of GA searches and to observe the effects of the different parameter configurations. In hindsight, we know that in practice this needs about $3.76 * 10^9$ fitness calculations. For a single core CPU algorithm, the estimated run-time for the whole examination is about 28.7 years. This is obviously unacceptable.

To speed-up this process, a graphics accelerator based implementation has been designed. Nowadays, Graphics Processing Units (GPUs) are highly paralleled devices containing thousands of processing elements and applicable for general purpose numerical computing. This leads to enormous processing power, but it is required to adapt the already existing sequential algorithms to massively parallel ones to utilise these resources fully.

It is easy to see that the proposed method is well-parallelizable and applicable for adaptation to a data-parallel fashion. The DHCP algorithm solves the same differential equation for all finite elements of the grid (with different input data). This data-parallel fashion is ideal for GPU implementation: one GPU thread is assigned to one element of the finite grid, and it is responsible for computing the temperature changes for this referenced location. In the case of a 10×34 sized grid ($n$ = 10, $m$ = 34 because of optimisation reasons), this needs 340 threads running the same function to calculate the heat movement for a given time interval.

## 3.3 Higher Level Parallelization

Running 340 threads on a modern GPU is not enough to fully utilise its processing power. In the case of NVidia GTX Titan Black cards, the number of cores is 2880. Thereby, executing 340 parallel threads leads only to very low (used cores / all cores = 340 / 2880 = 11.8%) theoretical occupancy (the practical utilisation is even worse).

It is worth noting that the proposed DHCP algorithm is a part of the fitness calculation process. During the GA, it is necessary to calculate the fitness for all chromosomes at the end of all iterations. In the case of 100 or more individuals, the number of parallel threads becomes 340x1000 = 34000 or more. This is enough parallelism to design and implement an efficient GPU-based implementation. We used the CUDA framework [15] and NVidia graphics cards for this purpose.

As a result of further optimisations, a novel data-parallel algorithm had been developed [16] with multi-GPU support [17] to speed-up the fitness calculations, and it is also possible to use all GPU devices and CPU cores together. By using two GPUs and four CPU cores, the run-time is about 100× less than the run-time of the original sequential method.

This implementation makes it possible to run thousands of GA searches within a reasonable period (14.5 weeks instead of 28.7 years) to evaluate all configurations to be examined.

# 4 Experimental Results

This paper follows the following terminology:

- Heat transfer simulation – using the DHCP solver to generate the 2D temperature history ($T_k^c$) based on the given parameters (HTC, material, and cooling medium attributes).
- Fitness calculation – calculating the fitness value ($F$) for a chromosome encoding an HTC. This takes the following steps: 1) running a heat transfer simulation using the given HTC values; 2) comparing the resulting thermal history to the reference curve.
- Iteration – one iteration of the GA. This takes the following steps: 1) selection; 2) crossover; 3) mutation; 4) fitness calculation for each chromosome.
- Search – the execution of a full GA process using a given configuration (population size, mutation rate, stopping condition). Main steps: 1) create initial population; 2) execute iterations; 3) stop the execution if one of the stopping conditions becomes true.
- Session – run several searches using the same configuration and evaluate the results (best-achieved fitness, an average of best fitnesses of all searches, average iteration count).
- Experiment – run several sessions using different configurations. 1st experiment: population size test; 2nd experiment: mutation probability test; 3rd experiment: stopping condition test.

## 4.1 Methodology

The estimation of the optimal population size, mutation probability and stopping condition for the proposed GA solving the IHCP is outlined. Due to the random behaviour of the genetic algorithms, it is difficult to define the most efficient configuration (the consecutive searches launched with the same parameters would give different results).

Several studies focusing on the evaluation of heuristic search algorithms suggest empirical validation of the parameters [10], [18]–[20]. According to the random behaviour of heuristics, it is usually not enough to run one search per configuration. It is necessary to execute as many examinations as possible and gather the following data:

- Best fitness value – the best-achieved fitness of the entire population;

- Number of iterations – as a secondary objective, we would like to find a parameter set, which is not just accurate but also fast; therefore, the number of iterations taken by the GA is important.
- Number of fitness function calculations – the number of iterations do not determine the required run-time. It is mostly based on the number of fitness function calculations (FFC), which is the multiplication of iteration count and population size.

Unlike many papers [20], we are not dealing with direct run-time. It is an architecture specific measure, and it is very hard to compare the results of different systems. We prefer the analysis of the number of FFC requirements because this is by far the most resource intensive part of the algorithm. The cost of one fitness function evaluation (running the DHCP solver) is independent of the actual HTC values. Therefore, the comparison of the number of these function calls is a good substitution for a platform independent run-time analysis.

The FFC count is also important to determine the end of a session testing a given configuration. It is necessary to run many simulations using the same configuration, but it is essential to set a limit for the number of these. It is common to set up a run-time limit, but as mentioned before it is unstable and platform specific. It is also unfair to limit the number of iterations because GAs with small population size needs fewer computations per iterations.

As a solution, the FFC count becomes the limiting factor. Every session testing a given configuration has a limit for FFC number (actually 50,000,000 calculations per session). The testing framework starts new GA searches one after the other monitoring the number of FFC count. When this accumulated number exceeds the predetermined limit, it starts the next session using the next configuration.

In the literature, there is no consensus on how to evaluate the efficiency of heuristics. Some papers [21] deal only the best fitness values found by a given configuration. Obviously, smaller fitness values mean better results, but according to the random behaviour of GAs, the comparison is based only on the best results found in all configurations is unsatisfactory. It does not give a clear estimation of the expected future performance. We followed the methodology of several papers [12], [22] comparing the average fitness values found by all GA searches of a session. Where required, we performed statistical tests to analyse the raw results.

## 4.2 Experimental Results

### 4.2.1 Population Size

We ran several sessions using the following parameters:

- Population size: 100, 200, 300, …, 2000

- Mutation rate: $M_L$=0.l; $R_L$=0.001;$M_M$=0.01;$R_M$=0.01;$M_S$=0.001;$R_S$=0.1
- Stopping condition: $STOP_{iter}$ = 30; $STOP_{rate}$ = 0.01

Table 1 and Fig. 1 show the experimental results.

Table 1

Best and average fitness values and the average iteration count of GAs with given population size. FFC count is equal to population × iteration.

| Population | Best fitness | Avg fitness | Avg iteration | Avg FFC count |
|---|---|---|---|---|
| 100 | 1936.2 | 4162.3 | 190 | 19000 |
| 200 | 699.4 | 5339.3 | 510 | 101959 |
| 300 | 480.5 | 4289.6 | 733 | 219982 |
| 400 | 259.3 | 3659.0 | 937 | 374824 |
| 500 | 307.8 | 3328.3 | 976 | 487932 |
| 600 | 239.1 | 520.2 | 1792 | 1075340 |
| 700 | 296.5 | 434.2 | 1927 | 1349176 |
| 800 | 249.8 | 374.3 | 1891 | 1513153 |
| 900 | 255.9 | 356.2 | 1798 | 1618403 |
| 1000 | 223.7 | 311.3 | 1859 | 1859185 |
| 1100 | 229.4 | 311.8 | 1846 | 2030116 |
| 1200 | 221.8 | 285.7 | 1807 | 2168800 |
| 1300 | 236.2 | 301.5 | 1745 | 2268795 |
| 1400 | 226.8 | 288.5 | 1689 | 2365236 |
| 1500 | 207.0 | 264.4 | 1708 | 2562000 |
| 1600 | 216.7 | 262.7 | 1663 | 2660716 |
| 1700 | 164.5 | 254.7 | 1655 | 2813122 |
| 1800 | 221.1 | 252.9 | 1621 | 2916900 |
| 1900 | 203.9 | 249.4 | 1680 | 3191169 |
| 2000 | 225.1 | 262.0 | 1578 | 3156750 |

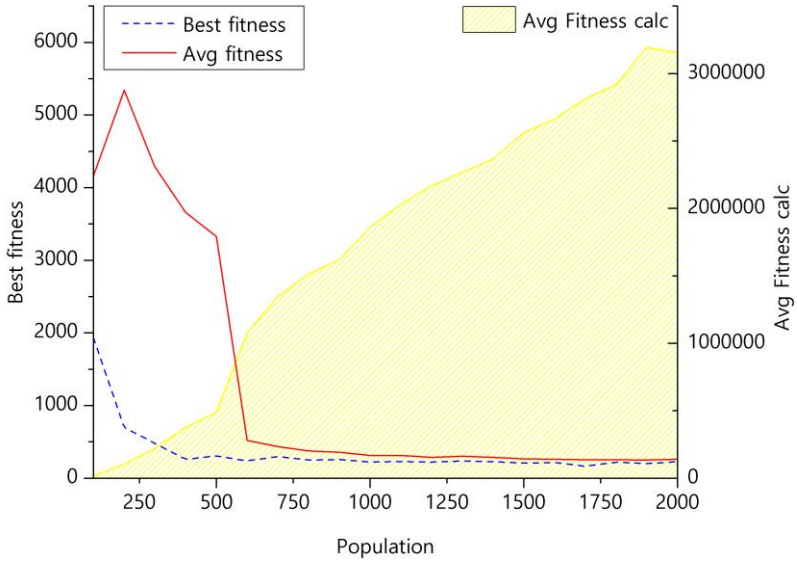


Figure 1

The solid red line shows the average fitness values, and the dashed blue one shows the best fitness values of GAs with the given population size. The yellow area shows the average FFC count.

In the case of small population sizes, the results are not satisfactory. There are not enough chromosomes to ensure convergence to a valid solution. These simulations are usually stopped in an early phase (sometimes without any progress). As can be seen in Fig. 1, where there is a significant improvement near population size 600. From 700 to 1500 the results become even better, but this trend slows down. Both the best fitness and the average fitness values become similar for larger population sizes.

The average number of necessary iterations is decreasing, but because of the higher population size, the number of FFCs (which requires the most computation effort) is increasing. Our first priority is accuracy, but efficiency is also important. Therefore, we should find the point with the smallest population size (and computation effort) after which there is no significant accuracy improvement.

Because of the spread of the results, the naïve comparison of the best fitness and average fitness values are not sufficient. One-way ANOVA tests were run using $\alpha$=0.05 significance level on the results of all searches [22], [23]. The null hypotheses was that the expected value of the average fitness is the same for all population sizes between P and 2000. In the case of small P values ($P \leq 1300$), it is obviously not true. When P=1400, the result of the ANOVA is: F=2.56 while $F_{critial}$=2.17; therefore, we have to reject the null hypotheses ($F > F_{critial}$) weakly. However, in the case of P=1500: F=0.55 and $F_{critial}$=2.30, the test shows ($F < F_{critial}$) that it is very likely that the expected value of the average fitness is the

same for both groups. In the case of larger P values, the results are similar. Thus, we can state that it is not worthwhile using a larger population size than 1500 because these sessions do not give a significant increase in accuracy, but have higher computational demand.

In the literature, it is also common to use two-tailed t-tests to compare the results of different parameter sets [21], [24]. By using these tests, the final verdict is the same. In comparing the results of the simulation with population size 1400 to the simulation using 2000 chromosomes, the t-test indicated some differences between the expected value of average fitness ($t_{stat}$ = 2.085 and $t_{critical}$=2.028, $t_{stat}$ > $t_{critical}$ - therefore, we have to reject the null hypothesis). Nonetheless, in the case of P=1500 ($t_{stat}$ = 0.184 and $t_{critical}$=2.032, $t_{stat}$ < $t_{critical}$) and larger P values, the t-tests show that the expected average fitness is the same as for population size 2000.

Based on these, our recommendation for population size is $\left| P \right|$=1500.

### 4.2.2 Mutation Probabilities

The purpose of the second experiment was to find the optimal mutation probabilities. Several sessions were run using the following parameters:

- Population size: 1500 (based on the result of the 1st experiment);
- Mutation rates: $M_L$=p/1000; $R_L$=0.01; $M_M$=p/2000; $R_M$=0.05; $M_S$=p/10000; $R_S$=0.25; where p = 0, 1, 2, … 9;
- Stopping condition: $STOP_{iter}$ = 30; $STOP_{rate}$ = 0.01.

In the case of GAs, it is always required to find the proper balance between exploration and exploitation ability of the search algorithm. Mutation is mostly responsible for the exploration part.

Table 2

Best and average fitness values and the average iteration count of GAs with given mutation probability. FFC count is equal to population × iteration.

| p value | Best fitness | Avg fitness | Avg Iteration | Avg FFC count |
|---|---|---|---|---|
| 0 | 534.0 | 772.7 | 292 | 438491 |
| 1 | 134.7 | 171.3 | 1918 | 2877333 |
| 2 | 119.0 | 152.9 | 2114 | 3170719 |
| 3 | 125.6 | 151.6 | 1988 | 2981559 |
| 4 | 122.9 | 149.6 | 1991 | 2985882 |
| 5 | 124.7 | 152.5 | 1862 | 2792417 |
| 6 | 127.7 | 155.0 | 1851 | 2777000 |
| 7 | 117.5 | 158.3 | 1758 | 2637632 |
| 8 | 137.9 | 158.6 | 1710 | 2565225 |
| 9 | 129.1 | 155.3 | 1686 | 2528775 |

The experimental results (Table 2) show the expected behaviour based on the literature. In the case of too small mutation rates, the exploitation failed. Chromosomes cannot get away from a local optimum. In the opposite case, too large mutation rates caused an "over randomised" search. These searches were more like a random search than a well-balanced GA. Both extremes led to poor performance.

Due to its nature, the analysis of the optimal mutation rate is simple compared to the population size analysis: p=4 gives the best average fitness values, and both lower and higher mutation rates give worse results.

According to this, the recommended mutation rate is $M_L$=0.004; $R_L$=0.01; $M_M$=0.002; $R_M$=0.05; $M_S$=0.0004; $R_S$=0.25.

### 4.2.3 Stopping Criteria

The primary aim was to find the parameter set which gives the highest accuracy (lowest fitness value). As an effect of the used elitism technique, the fitness value is monotonically decreasing during the search. Therefore, it is evident that if only accuracy is taken into account, it is worth running the algorithm as long as possible.

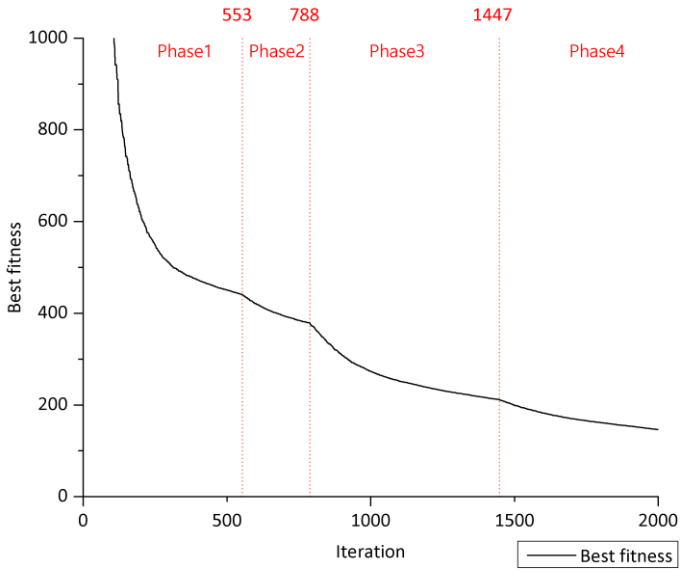


Figure 2

Best fitness values by iterations for a given GA search. Vertical lines show the phase switching points. It is visible that all phases have the same characteristic.

However, a novel technique dynamically changing the number of control points is also used. Fig. 2 shows the fitness values by iteration number for a given GA execution. The common characteristic of these phases is well visible: every phase starts with a fast decreasing period and some iterations later this slows down. This raises an additional question. At which point is it worth stepping over to the next phase (and using more control points)?

If this happens too late, the full GA convergence becomes slow. Early phases use only a few control points, which has a significant limitation in describing the HTC function. When the GA reaches this limit, the decrease of the fitness value becomes almost negligible. It is not worth leaving the algorithm to fine-tune these results because it would be more efficient to change to the next phase which has fewer limitations.

It is also worth avoiding the opposite situation and switching to the next phases too early. First, notice that this dynamic resolution model is an essential part of the algorithm. Without this ($STOP_{iter}$=0 in Table 3) the GA cannot start to converge (in the case of 340 starting parameters, the search space is too large). Too early phase switching leads to similar problems: the algorithm will not be able to converge ($STOP_{iter}$=1 in Table 3), or it converges, but starts the last phase with a relatively poor fitness value, and it takes many iterations to improve this.

Table 3

Fitness values after the given number of iterations using different $STOP_{iter}$ parameters. The first test ($STOP_{iter}$=0) does not use the dynamic resolution method.

| **$STOP_{iter}$** | **10000** | **20000** | **30000** | **40000** | **50000** | **60000** |
|---|---|---|---|---|---|---|
| 0 | 1855.68 | 1619.75 | 1444.06 | 1318.32 | 1217.46 | 1122.83 |
| 1 | 4375.95 | 4002.08 | 3798.09 | 3616.72 | 3475.45 | 3276.50 |
| 10 | 67.25 | 61.74 | 59.76 | 58.66 | 57.71 | 57.09 |
| 50 | 64.72 | 59.31 | 57.38 | 56.29 | 55.66 | 55.25 |
| 100 | 62.10 | 57.30 | 55.44 | 54.39 | 53.76 | 53.38 |
| 200 | 108.40 | 53.12 | 51.33 | 50.40 | 49.84 | 49.31 |
| 300 | 213.28 | 52.95 | 50.95 | 50.00 | 49.18 | 48.65 |
| 500 | 213.71 | 86.29 | 51.69 | 49.12 | 48.19 | 47.60 |
| 600 | 212.72 | 100.53 | 53.37 | 48.38 | 47.28 | 46.61 |
| 700 | 222.54 | 122.04 | 63.93 | 48.87 | 47.86 | 47.23 |
| 900 | 221.47 | 173.51 | 83.78 | 53.30 | 48.61 | 47.40 |
| 1000 | 228.76 | 174.47 | 85.74 | 52.03 | 48.83 | 47.77 |

Table 3 shows the experimental results. The GA had to finish all phases (except the last one) when the improvement of the best fitness value was less than 1% ($STOP_{rate}$=0.01) in the last $STOP_{iter}$ iterations. There was no similar limitation for the final phase, and the GA was left running for 60000 iterations.

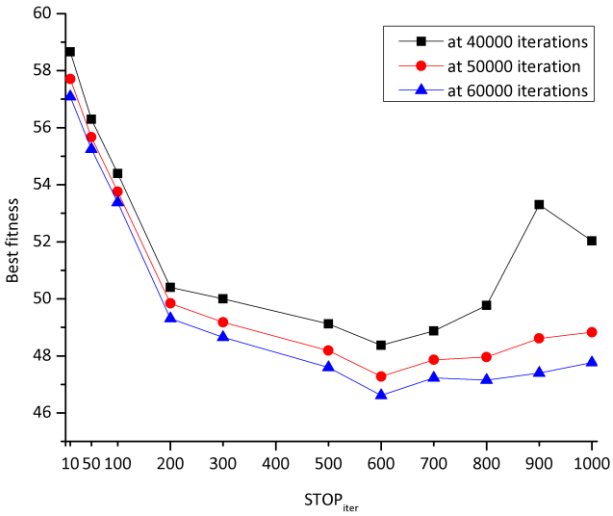


Figure 3

Best achieved fitness values by $STOP_{iter}$ parameter values after given iteration numbers. Blue triangles show the final results (after 60000 iterations), black rectangles show a middle-state after 40000 iterations, red circles show the state between them (50000 iterations).

As is visible from the table and Fig. 3, the optimal value was 600. Lower parameters give poor final fitness. A slight weakening is also visible in the case of higher numbers. It is not significant, but it is worthwhile seeing that without this increment (and considering the fitness values identical for these parameters), we should recommend the same because of the faster convergence. The secondary objective is to find the fastest parameters, and it is visible that GA with $STOP_{iter}$=600 finds the best fitness values earlier.

Based on the results, the recommended stopping criteria is $STOP_{iter}$=600.

## Conclusions

Using GA to solve the IHCP is already a known procedure. Nevertheless, the best configuration parameters were unknown to use this method efficiently, and the high computation demand makes it impossible to determine these using experimental test.

The determination of GA's configuration parameters providing the highest efficiency to solve a 2D axis-symmetrical IHCP problem is outlined. A computational framework was developed by which thousands of GA searches have been analysed and the performance of several configurations (population size, mutation probability, stopping condition) has been evaluated.

As a final result, the following recommendations are made:

- Population size: $|P|$=1500

- Mutation rate: $M_L$=0.004; $R_L$=0.01; $M_M$=0.002; $R_M$=0.05; $M_S$=0.0004; $R_S$=0.25
- Stopping condition: $STOP_{iter}$=600

Using these parameters, stable and efficient GA searches can be expected without wasting resources for unnecessary computations.

The most advanced part of the framework is the hybrid (CPU and GPU) parallel DHCP solver module. One of the design concerns was the ability to use this module with other search methods, like PSO, or Fireworks. As a further plan, a request to extend the research with the investigation of these heuristics is made.

It is also possible to improve the efficiency of the already existing DHCP solver. Using more than two graphics cards can linearly improve the computing performance, and the new NVLINK™ technology developed by NVIDIA provides improved GPU-to-GPU link bandwidth and tight integration with IBM Power CPU makes it possible to decrease the memory transfer deficits significantly.

**Acknowledgements**

We acknowledge the financial support of this work by the Hungarian State and the European Union under the EFOP-3.6.1-16-2016-00010 project and Hungarian-Japanese bilateral Scientific and Technological (TÉT_16-1-2016-0190) project. The authors would like to thank NVIDIA Corporation for providing graphics hardware for the GPU benchmarks through the CUDA Teaching Center program. We also would like to thank IBM Systems for the temporary access to their IBM NVIDIA Acceleration Lab which allows us to run some additional experiments using their new NVLink™ capable Power system.

Supported BY the ÚNKP-17-4/I. New National Excellence Program of the Ministry of Human Capacities.

**References**

[1] P. Oksman, S. Yu, H. Kytönen, and S. Louhenkilpi, “The Effective Thermal Conductivity Method in Continuous Casting of Steel,” *Acta Polytech. Hungarica*, vol. 11, no. 9, 2014.

[2] O. M. Alifanov, *Inverse Heat Transfer Problems*. Springer, 1994.

[3] J. V. Beck, B. Blackwell, and C. R. J. St. Clair, *Inverse Heat Conduction*. New York: Wiley, 1985.

[4] M. J. Colaço, H. R. B. Orlande, and G. S. Dulikravich, “Inverse and optimization problems in heat transfer,” *J. Brazilian Soc. Mech. Sci. Eng.*, vol. 28, no. 1, pp. 1–24, 2006.

[5] M. N. Özisik and H. R. B. Orlande, *Inverse Heat Transfer: Fundamentals and Applications*. Taylor & Francis, 2000.

[6] I. Felde, *Estimation of thermal boundary conditions by gradient based and genetic algorythms*, vol. 729. Trans Tech Publications, 2013.

[7] S. Vakili and M. S. Gadala, "Effectiveness and Efficiency of Particle Swarm Optimization Technique in Inverse Heat Conduction Analysis," *Numer. HEAT Transf. PART B-FUNDAMENTALS*, vol. 56, no. 2, pp. 119–141, 2009.

[8] I. Felde, S. Szénási, A. Kenéz, S. Wei, and R. Colas, "Determination of complex thermal boundary conditions using a Particle Swarm Optimization method," in *5th International Conference on Distortion Engineering*, 2015, pp. 227–236.

[9] I. Felde and S. Szénási, "Estimation of temporospatial boundary conditions using a particle swarm optimisation technique," *Int. J. Microstruct. Mater. Prop.*, vol. 11, no. 3/4, pp. 288–300, 2016.

[10] M. A. Panduro, C. A. Brizuela, L. I. Balderas, and D. A. Acosta, "A comparison of genetic algorithms, particle swarm optimization and the differential evolution method for the design of scannable circular antenna arrays," *Prog. Electromagn. Res. B*, vol. 13, pp. 171–186, 2009.

[11] D. E. Goldberg and M. Rudnick, "Genetic Algorithms and the Variance of Fitness," *Illinois Genet. Algorithms Lab. Rep.*, vol. 5, no. 91001, pp. 265–278, 1991.

[12] Y. R. Tsoy, "The influence of population size and search time limit on genetic algorithm," *Sci. Technol. 2003. Proc. KORUS 2003. 7th Korea-Russia Int. Symp. (Volume3 )*, no. 1, pp. 181–187, 2003.

[13] M. Mitchell, "An introduction to genetic algorithms," *Comput. Math. with Appl.*, vol. 32, no. 6, p. 133, 1996.

[14] G. Syswerda, "Uniform Crossover in Genetic Algorithms," in *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989, pp. 2–9.

[15] NVIDIA, "CUDA C Programming Guide." 2014.

[16] S. Szénási, I. Felde, and I. Kovács, "Solving One-dimensional IHCP with Particle Swarm Optimization using Graphics Accelerators," in *10th Jubilee IEEE International Symposium on Applied Computational Intelligence and Informatics*, 2015, pp. 365–369.

[17] S. Szénási and I. Felde, "Heat Transfer Simulation using GPUs," in *20th IEEE Jubilee International Conference on Intelligent Engineering Systems*, 2016, pp. 263–267.

[18] T. Weise, Y. Wu, R. Chiong, K. Tang, and J. Lässig, "Global versus local search: the impact of population sizes on evolutionary algorithm

performance," *J. Glob. Optim.*, no. March, pp. 1–24, 2016.

[19] K. Mills, J. Filliben, and A. Haines, "Determining relative importance and effective settings for genetic algorithm control parameters," *Evol. Comput.*, no. x, 2015.

[20] R. H. Abiyev and M. Tunay, "Experimental Study of Specific Benchmarking Functions for Modified Monkey Algorithm," *Procedia Comput. Sci.*, vol. 102, no. August, pp. 595–602, 2016.

[21] D. Whitley, S. Rana, and R. B. Heckendorn, "The island model genetic algorithm: On separability, population size and convergence," *J. Comput. Inf. Technol.*, vol. 7, pp. 33–47, 1999.

[22] I. Rojas, J. González, H. Pomares, J. J. Merelo, P. a. Castillo, and G. Romero, "Statistical Analysis of the Main Parameters Involved in the Design of a Genetic Algorithm," *Syst. Man, Cybern. Part C Appl. Rev. IEEE Trans.*, vol. 32, no. 1, pp. 31–37, 2002.

[23] P. A. Castillo-Valdivieso, J. J. Merelo, A. Prieto, I. Rojas, and G. Romero, "Statistical analysis of the parameters of a neuro-genetic algorithm," *IEEE Trans. Neural Networks*, vol. 13, no. 6, pp. 1374–1394, 2002.

[24] A. Silva, A. Neves, and E. Costa, "An empirical comparison of particle swarm and predator prey optimisation," *Artif. Intell. Cogn. Sci.*, pp. 1–45, 2002.