Modified Particle Swarm Optimization Method to Solve One-dimensional IHCP

Sándor Szénási
John von Neumann
Faculty of Informatics
Óbuda University
Budapest, Hungary
Email: szenasi.sandor@nik.uni-obuda.hu

Imre Felde
John von Neumann
Faculty of Informatics
Óbuda University
Budapest, Hungary
Email: felde.imre@nik.uni-obuda.hu

***Abstract*—Particle swarm optimization is one of the most promising methods for solving the inverse heat conduction problem. However, it needs a high computing capacity and a long time to find optimal solutions for large scale problems (large swarm populations). This paper presents a further improvement of our already published GPU-based method. Our new method extends the original algorithm with an initial temporary swarm (which contains significantly more participants than the simulated swarm to better cover the search space). The results are promising, the average fitness values are usually better when using this technique, and (thanks to the well-parallelized graphics accelerator based implementation) the runtime is quite similar.**

I. Introduction

A heat transfer process simulation requires functions and parameters that describe the real heat transfer phenomena obtained during cooling. Typical quantities that characterize the heat extraction ability of a cooling medium are the heat transfer coefficient (HTC) and heat flux (HF). During the quenching in liquids, generating vapour and boiling stages (i.e., oils, water, polymer solutions) the heat transfer is altered significantly depending on its surface temperature. In addition, heat transfer is not uniform across the whole surface domain of the workpiece. It follows that a realistic simulation of quenching is assumed to apply thermal boundary conditions (HTC or HF) as functions of surface temperature and local coordinates.

To solve the inverse heat conduction problem (IHCP) [1], there are several implicit and explicit formulations. In the first approach, the problem is formulated as a multivariable optimization problem [2]–[4]; the latter method attempts to directly determine the unknown parameters using regularization techniques to solve the resulting system of equations [5]–[7]. In this paper, we use a stochastic approach that is based on particle swarm optimization (PSO).

Particle swarm optimization was developed in 1995 by Eberhardt and Kennedy [8], [9]. It is a biologically-inspired search and optimization method based on the behaviours of birds flocking or fish schooling. The main idea of the algorithm is the simulation of animal groups without any distinguished leader (such as birds and fishes). The communication between these group members is not one-to-many but many-to-many: the position and speed of each member has effect on the others.

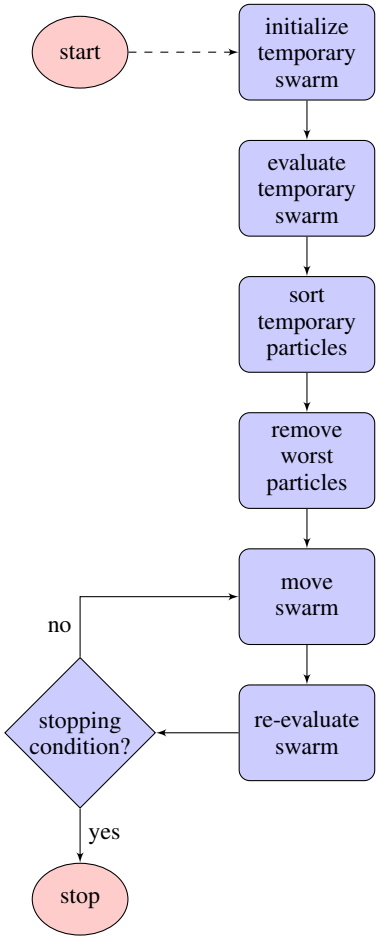

Fig. 1: Modified Particle swarm optimization method

There is no predetermined hierarchy between the members of the group; the strength of the aforementioned effect is based on the distance from the target (food in biological examples). The members of the flock follow the other members having the closest position to the target.

We can use similar techniques for general optimization tasks. We have several group members (each particle represents a possible solution) in different locations (the parameters

978-1-4673-8520-6/15/$31.00©2015 IEEE

of the solution are coded as a location). A function is required to evaluate the position of any particle (fitness function). Participants access the locations and velocities of all other members (however, it is usually enough to know the location of the current best potential solution) and can set a new velocity and location based on these.

There are several movement strategies; the actual problem determines which is best for a given purpose. We expect that (after some iterations) the location of the particle with highest fitness value will show the optimal solution (however, this is not guaranteed, and the method is sensible for local optimums).

II. Methodology

There are several modifications from the original particle swarm optimization [10]. The goal of these is (usually) to speed up calculations and increase convergence.

In the case of an HTC application [12], [13], we can speed up the calculation using graphics accelerators [11]. The convergence of the method is satisfactory; it usually needs about 100-200 iterations to stop and return the actual solution. However, we faced another issue, the PSO often finds a local optimum (but not the global one), because the parameter space is too large. In the case of 1D HTC applications, all particles are located in an 11-dimensional space; in the case of 2D calculations, this dimension is significantly higher (55). In a typical application, the swarm consists of 1000-10000 pieces of particles. Obviously, that it is not possible to cover the entire space with particles. Therefore, the result of an optimization depends strongly on its random starting positions [14]. In unlucky situations, none of the particles start near the global optimum; this usually leads to misleading final results.

Increasing the number of particles can partially solve this issue, but raises several other issues:

- The evaluation of a significantly higher number of particles requires a higher computing capacity and a larger runtime.
- Because of its random starting location generation, this way neither ensures a global optimum.

Our idea for solving this problem is to extend the original PSO algorithm by one additional step. In the initialization phase, the modified algorithm generates more particles than the common swarm size. In the next step, it evaluates all of these particles (this is possible because we can calculate the fitness function for all particles independently from the others). After these additional initial steps, we keep only the best particles; these form the initial swarm. This method has the following advantages:

- A significantly higher number of generated particles increase the coverage of the search space. This increases the probability of placing a particle near the global optimum.
- The evaluation of this temporary swarm is highly compute intensive; but we keep only the best particles for further processing. Therefore, in the further iterations, we can work with a smaller swarm.

III. Evaluation

We ran several tests to evaluate this new method. The input heat function was the same for all tests, but the number of particles was different. The swarm size was given by the following two parameters:

- N – common swarm size (as used in the original PSO).
- M – size of the (temporary) initial swarm.

The tests were based on the following common swarm sizes (N): 10, 20, 40, 80, 160 particles. The temporary initial swarm size (M) was the multiplication of the common swarm size: 1x, 2x, .., 14x, 15x. The particles of the first generation were randomly generated. The stopping condition for the simulation was the following in all tests: if the best global fitness value of the swarm was the same for at least 10 steps, the simulation stops.

We used the following hardware configurations for the tests:

- CPU configuration
  - Processor: Intel® Core™ i7-2600
  - Architecture: Sandy Bridge
  - Number of cores: 4
  - Memory: 16GB DDR2
- GPU configuration
  - Graphics accelerator: NVIDIA Tesla K40c
  - Architecture: Kepler (GK110B)
  - Number of shaders: 2880
  - SMX Count: 15
  - Memory: 12GB GDDR5

Due to the stochastic behaviour of the PSO algorithm, it is not enough to run one test for each parameter sets. To decrease uncertainty, we ran 100 tests for all parameter sets and calculated the average of these results.

The following values were monitored:

- The fitness value of the best particle at the end of the simulation. The result of the fitness function was the difference between the reference heat function (cooling curve), and the heat function derived from the particle parameters (therefore, smaller values are better).
- The number of iterations. Better convergence means fewer iterations. Smaller values mean less time and computational requirements; therefore, these are better.
- The runtime of the fitness evaluation of the initial temporary swarm.
- The aggregate runtime of the fitness evaluation of the further iterations of the real swarm.

Table I shows the final results of the 5600 executed tests (4x14x100). *M* is the size of the initial swarm; *N* is the size of the swarm in the further iterations; *best fitness* is the average of the best fitness values of the best participants at the end of the simulation; *initial swarm evaluation time* is the average requested time for evaluating the first (larger) swarm (ms); and *aggregated evaluation time* is the average requested time for the evaluation of all following iterations (ms). As it is visible, in the first line, the size of the initial swarm is the same as

TABLE I: Test results

M	N	Iter.	Best fitness	Initial swarm eval. time (ms)	Aggregated eval. time (ms)
10	10	122	893	613	26246
20	10	124	957	631	26792
30	10	119	853	662	25674
40	10	121	860	664	26107
50	10	125	907	670	26972
60	10	123	873	672	26569
70	10	125	793	696	26864
80	10	116	849	661	25116
90	10	120	803	656	25939
100	10	125	705	664	26747
110	10	123	875	667	26555
120	10	117	777	672	25307
130	10	120	843	674	25970
140	10	123	853	676	26772
20	20	107	689	609	23261
40	20	103	793	621	22544
60	20	105	712	638	23083
80	20	99	730	654	21931
100	20	106	707	667	23261
120	20	107	652	675	23463
140	20	98	753	678	21669
160	20	108	775	679	23828
180	20	104	660	678	22818
200	20	106	684	681	23233
220	20	107	813	683	23594
240	20	106	788	686	23485
260	20	106	669	691	23417
280	20	106	656	690	23448
40	40	100	571	626	21948
80	40	99	655	651	21925
120	40	99	665	671	22010
160	40	101	636	677	22336
200	40	104	620	681	22941
240	40	99	685	685	22236
280	40	101	596	691	22471
320	40	101	628	692	22323
360	40	105	599	695	23297
400	40	101	599	702	22480
440	40	102	596	704	22647
480	40	102	619	705	22608
520	40	100	589	705	22276
560	40	105	629	706	23241
80	80	97	599	652	22842
160	80	102	571	680	24088
240	80	96	587	686	22808
320	80	97	573	694	22895
400	80	94	560	701	22222
480	80	102	551	703	24002
560	80	96	636	710	22865
640	80	100	558	717	23509
720	80	104	558	719	24593
800	80	101	588	730	23832
880	80	100	598	730	23641
960	80	101	600	734	23905
1040	80	99	562	743	23598
1120	80	99	577	743	23574
160	160	96	546	685	23347
320	160	94	540	700	22853
480	160	94	539	711	22982
640	160	97	562	721	23639
800	160	98	540	732	23970
960	160	97	543	738	23723
1120	160	99	540	749	24116
1280	160	96	547	757	23530
1440	160	98	542	769	23995
1600	160	96	541	778	23656
1760	160	93	548	782	23129
1920	160	97	535	791	23906
2080	160	96	538	844	23986
2240	160	99	532	858	24659

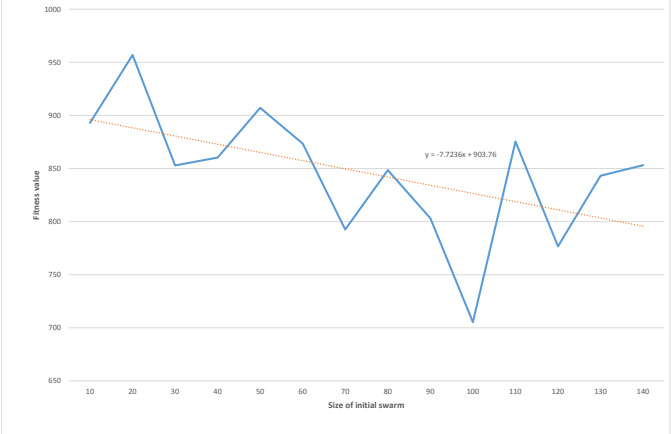

Fig. 2: Fitness values based on the size of the initial swarm in the case of common swarm size 10

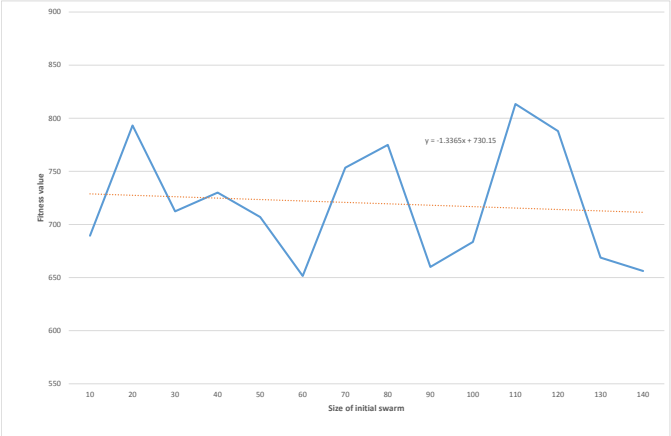

Fig. 3: Fitness values based on the size of the initial swarm in the case of common swarm size 20

the size of the swarm in the further iterations. Therefore, this can be considered as the original algorithm.

Figure 2–6 show average fitness values in cases of different swarm sizes. These fitness values change hectically, but the applied trend lines show that (in all cases) a larger number of initial temporary particles lead to better results.

Evaluation of the further results shows the requested number of steps is roughly the same. Therefore, the convergence of the PSO is not changed. More particles in the first generation require more calculation efforts; this leads to higher runtime, but (thanks to the parallel GPU implementation) is not a real issue. The runtime of the remaining simulation is (obviously) the same as in the case of the original algorithm, because the swarm size is the same.

IV. Conclusions

This paper presents the evaluation of a modified particle swarm optimization algorithm to solve the inverse heat transfer problem. Our assumption was that a larger initial swarm can

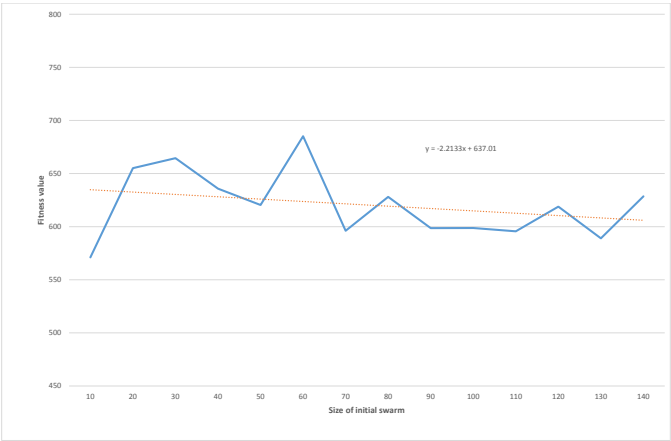

Fig. 4: Fitness values based on the size of the initial swarm in the case of common swarm size 40

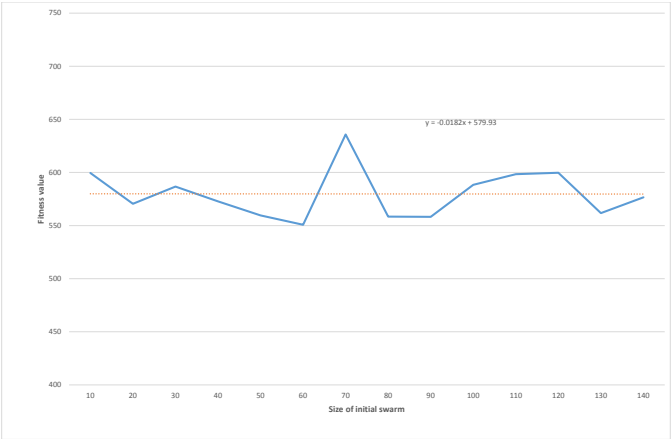

Fig. 5: Fitness values based on the size of the initial swarm in the case of common swarm size 80

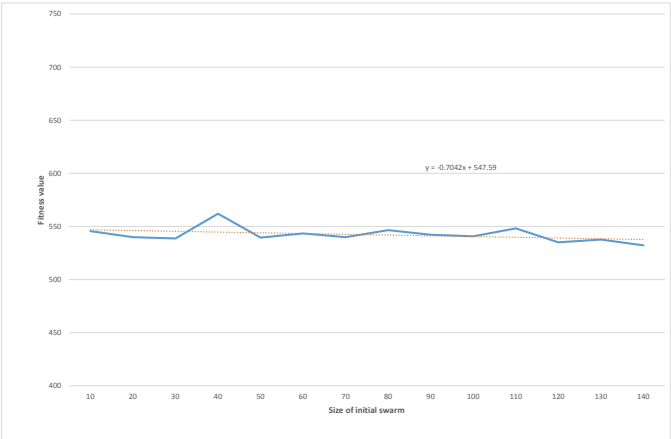

Fig. 6: Fitness values based on the size of the initial swarm in the case of common swarm size 160

cover a larger part of the search space and lead to better fitness values.

As is visible in the results, this has been confirmed. Larger initial swarms (followed by smaller swarms in a real simulation) lead to better fitness values. The runtime of the evaluation of the initial swarm is higher than in the case of the original algorithm, but this disadvantage does not significantly affect overall runtime.

Acknowledgements

We acknowledge the financial support of this work by the Hungarian State and the European Union under the TÁMOP-4.2.2/A-11/1-KONV-2012-0029 and the Chinese-Hungarian S&T bilateral project TÉT_12_CN-1-2012-0027 projects. The authors would like to thank NVIDIA Corporation for providing graphics hardware for the GPU benchmarks through the CUDA Teaching Center program. The authors also wish to thank the members of the CUDA Teaching Center at Óbuda University for their constructive comments and suggestions. The content of the paper however does not necessarily express the views of these companies and persons, the authors take full responsibility for any errors or omissions.

References

[1] I. Felde and W. Shi, "Hybrid approach for solution of inverse heat conduction problems," in *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, 2014,

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

[3] M. N. Özisik and H. R. B. Orlande, *Inverse Heat Transfer: Fundamentals and Applications*.

[4] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313,

[5] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions," *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 112–147,

[6] R. Das, "A simplex search method for a conductive–convective fin with variable conductivity," *International Journal of Heat and Mass Transfer*, vol. 54, pp. 5001–5009,

[7] O. Nelles, *Nonlinear system identification*.

[8] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, pp. 1942–1948,

[9] D. Palupi Rini, S. Mariyam Shamsuddin, and S. Sophiyati Yuhaniz, "Particle Swarm Optimization: Technique, System and Challenges," *International Journal of Computer Applications*, vol. 14, no. 1, pp. 19–27,

[10] Y. del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J. C. Hernandez, and R. G. Harley, "Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems," *Evolutionary Computation, IEEE Transactions on*, vol. 12, no. 2, pp. 171–195,

[11] 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*. Timisoara, Romania: IEEE, 2015,

[12] A. Bogárdi-Mészöly, A. Rövid, H. Ishikawa, S. Yokoyama, and Z. Vámossy, "Tag and Topic Recommendation Systems," *Acta Polytechnica Hungarica*, vol. 10, no. 6, pp. 171–191,

[13] S. Sergyán and L. Csink, "Automatic Parametrization of Region Finding Algorithms in Gray Images," in *4th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, Timisoara, Romania, 2007,

[14] E. Campana and G. Fasano, "Dynamic system analysis and initial particles position in particle swarm optimization," *IEEE Swarm Intelligence*, no. 1,