DGPF

An Adaptable Framework for Distributed Multi-Objective Search Algorithms Applied to the Genetic Programming of Sensor Networks

Thomas Weise, Kurt Geihs
University of Kassel, Wilhelmshöher Allee 73, 34121 Kassel, Germany
{weise,geihs}@vs.uni-kassel.de

Abstract: We present DGPF, a framework providing multi-objective, auto-adaptive search algorithms with a focus on Genetic Programming. We first introduce a Common Search API, suitable to explore arbitrary problem spaces with different search algorithms. Using our implementation of Genetic Algorithms as an example, we elaborate on the distribution utilities of the framework which enable local, Master/Slave, Peer-To-Peer, and P2P/MS hybrid distributed search execution. We also discuss how heterogeneous searches consisting of multiple, cooperative search algorithms can be constructed. Sensor networks are distributed systems of nodes with scarce resources. We demonstrate how Genetic Programming based on our framework can be applied to create algorithms for sensor nodes that use these resources very efficiently.

Key words: Heuristic, Randomized, Search Algorithms, Distributed Genetic Algorithms, Genetic Programming, Auto-Adaptation, Sensor Networks, Sensor Nodes

1. INTRODUCTION

Find an election algorithm for a given sensor network with minimum instruction count which minimizes energy consumption due to transmissions. Construct the best aerodynamic shape of an airplane wing while maximizing its stability using the minimal amount of material. Many search algorithms

can be applied to solve such complex problems [1]. There exist artificial approaches like Tabu Search or randomized Hill Climbing, physically inspired ones like Simulated Annealing as well as methods of biological origin like Genetic Algorithms.

For most problems it is not a priori possible to decide which algorithm and parameter configuration will perform best. Practical experiences often apply to a certain problem only and cannot be generalized. To implement different search algorithms or to customize multiple search libraries however is normally costly and time-consuming.

The performance of search algorithms can often be improved by distributing the computational load to a network of computers. If parameters like the mutation-rate in Genetic Algorithms or the length of the Tabu-List in Tabu Search are adapted dynamically, the performance may increase further [2].

When performing Genetic Programming of real algorithms with its usually very rugged fitness landscapes, a search framework taking advantage of all these improvement options is needed.

In this paper we introduce a Common Search API of our DGPF framework [3],[4], allowing the evaluation of arbitrary problem spaces to be performed with different or even multiple cooperating, distributed, auto-adaptive search algorithms. We will furthermore show the utility of our framework for automated algorithm creation for sensor networks by evaluating an experiment.

2. RELATED WORK

In the past there have been successful applications of other search methodologies as back-ends of GA [14], [15], [16]. Meta-Heuristics like the one introduced by Talibi [17] already confirmed that the cooperation of different, hierarchical coupled search algorithms provides remarkable advantages. Yao has melted GA and Simulated Annealing together to create a new, improved version, the Genetic Annealing [19]. Our framework extends such ideas by integrating arbitrary search algorithms to cooperatively work together on one problem.

O'Reilly and Oppacher have suggested replacing GA as foundation for GP [13] with other heuristics like Simulated Annealing and Stochastic Iterated Hill Climbing. Applying such methods is simplified by our framework a lot. The GP layer or any other given Problem Space Implementation can rest on the Common Search API, which internally might run any search algorithm implemented.

Most of the research stated above does not concern multi-objective

optimization [1]. Our search API on the other hand provides building blocks which ease the construction of such algorithms.

A lot of work has been done on the self-adaptation of search algorithms [18], [2]. If a search heuristic is implemented using our framework, it will automatically be equipped with this ability too. It may use different strategies that can even be exchanged at runtime.

3. FRAMEWORK STRUCTURE

The core of our framework is formed by a Common Search API, which defines some classes and prototypes to be used. This API can be accessed from two sides (see Figure 1): On the one side different search algorithms can be implemented, providing the functionality needed to perform randomized heuristic searches.

The user, on the other side, has to implement the functionality needed to explore the problem space and, if needed, to simulate possible solutions. In the style of multi-objective Genetic Algorithms, she may use different fitness functions to evaluate the simulated solution candidates.

The user-defined code can then be used in conjunction with any search algorithm made available by the framework. Hence, a direct comparison and selection of the optimal approach for a given problem has become straightforward.

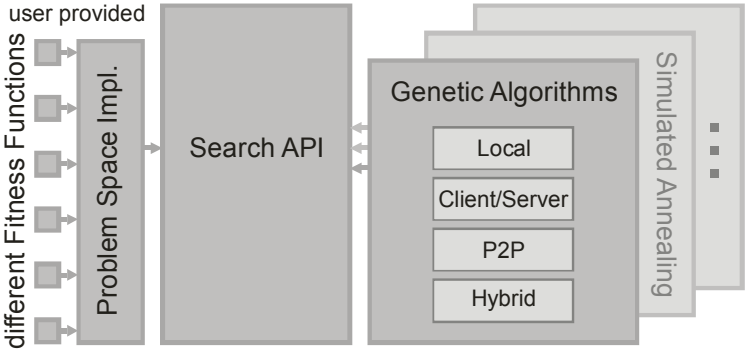

Figure 1. The structure of the DGP Framework's search abilities.

3.1 The Common Search API

The search API introduces four essential tools and abstractions: a finite state machine which governs the control loop shared by the search algorithms, means for the user to plug in fitness functions and problem domain specific functionality, basic auto-adaptation support, and distribution utilities.

Search algorithms in general, if bio-inspired, randomized, or otherwise heuristic, can be performed using the finite state machine presented in Figure 2 as control loop. They can be divided into single search steps representing for example a generation in Genetic Algorithms, or a state transition in Simulated Annealing. After each step the state information, for instance containing the best individual found in the search and its fitness values, will be updated. Status events will now be generated to inform the application using the search algorithm. To limit the runtime of the search, the user may provide certain thresholds, like a maximum search time, a maximum update count, optimum fitness values, and so on, in order to define when the search should be halted automatically. If these criteria are not met, the search parameters can auto-adapt to the new situation and the next step will be initiated.

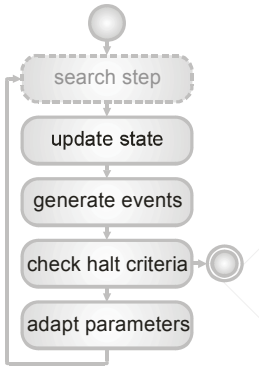

Figure 2. The control loop FSM.

To investigate a custom problem space, the user has to plug in the "Problem Space Implementation" (see Figure 1) which consists of three parts:

1. The type of individuals to be examined, which can be anything from simple numbers if optimizing a mathematical problem to complex construction plans for airplane wings.
2. The methods needed to randomly create initial individuals and to derive new individuals from either one or two already existing ones.
3. Means to simulate these individuals in order to check their fitness.

Based on this implementation the user can now define multiple fitness functions, regarding different functional and non-functional aspects of the individuals evaluated.

Apart from Genetic Programming for sensor networks, we exemplarily created a Problem Space Implementation for Semantic Web Service composition, able to solve problems like the WSC Challenge [6], as a proof of concept.

The Common Search API includes facilities for both parallelization and distribution which will be discussed in the next section.

3.2 Genetic Algorithms and the Distribution Schemes of the DGPF

The most popular biologically inspired search and optimization methods by far are Genetic Algorithms. Genetic Algorithms follow a well known schema which closely matches the search control loop FSM introduced in the

previous section. Starting with an initial population, the individuals are evaluated, statistical information is updated and individuals are selected for reproduction in the next iteration.

Distributed Genetic Algorithms outclass their locally running counterparts in many applications [7]. Let us thus discuss the distribution utilities of the Common Search API exemplarily for the DGPF implementation of GA.

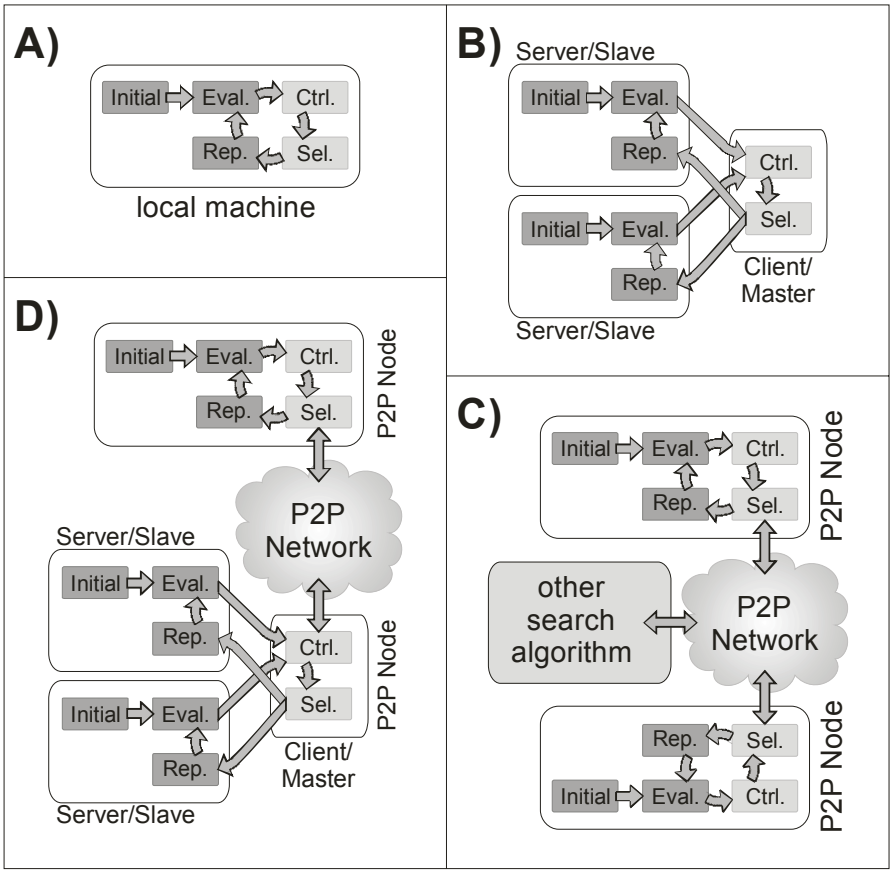

Figure 3. Different distribution schemes for GA provided by the DGPF.

Four different distribution forms of Genetic Algorithms are provided, as illustrated in Figure 3. The default method is to run a search locally (A). If more than one machine is available in a network, the tasks of creating and evaluating individuals can be distributed. This technique is called Client/Server or Master/Slave (B) approach [5], [20]. This is useful if the evaluation involves time-consuming simulations. If network bandwidth is limited or large populations are needed, a Peer-To-Peer approach should be

chosen (C) [7], [8]. Different machines running Genetic Algorithms are now able to cooperate using the Island Hopping paradigm. Last but no least, a hybrid distribution scheme (D) of mixing the Peer-To-Peer and Client/Server techniques allows different networks or clusters to cooperate on the same search.

The Client/Server- and the P2P-components are unified in the Common Search API. Therefore, they can be used by all search algorithms implemented in the DGPF, allowing even totally different algorithms like GA, Simulated Annealing, Tabu Search and Hill Climbing to be incorporated into one heterogeneous search.

The distribution methods discussed here are built using self-healing and error-tolerant techniques. Thus, a Client/Server system will continue its work even if all but one server are switched off by a hardware-reset. A search using the P2P-distribution will keep running even if all other P2P-nodes are shut down. If some of the other machines happen to be restarted, they will seamlessly be integrated into the search again by both technologies.

Other unified base structures of the DGPF are comparators, sorting and selection schemes. A comparator is used to determine which individuals are dominated by which other ones. The sorting schemes allow individuals to be sorted according to these comparators or by using additional statistical measures. Combined with the non-dominated individual list maintained by the API, multi-objective search algorithms can easily be constructed. As a combination of these four features, the NPG-Algorithm [10] has exemplarily been implemented.

4. GENETIC PROGRAMMING OF SENSOR NETWORKS

Today we experience a growing demand for distributed systems of sensors [9]. In this chapter, we describe how the DGPF framework is used to genetically create algorithms for such sensor networks.

Sensor nodes are small devices that gather sensor information about their environment and transmit it wirelessly. They are restricted in resources like memory size, processing speed, and, most important, battery power. The communication among them is not reliable and the topology of their network is volatile. The program code created for sensor nodes should thus be robust and as efficient as possible.

Our goal is the automated creation of algorithms for sensor nodes. We apply multi-objective Genetic Programming since it allows optimizing the algorithms created not only for functionality but also for the economical use of resources, especially for minimizing expensive communication.

To evaluate the fitness of such algorithms we simulate whole sensor networks. In our model, sensor nodes are represented as virtual machines with a fixed-sized memory architecture, asynchronous IO, and a Turing-complete instruction set ([11], [12]).

Many nodes (the virtual machines) run asynchronously in the simulation at approximately the same speed which, however, might differ from node to node and cannot be regarded as constant. The nodes are connected wirelessly and thus cannot a priori guarantee reliable communication. It is not possible to send directed transmissions. Like radio broadcasts they will be received by any node in range.

With such simulations we can transform global behavior of a network into local behavior of single nodes using GP.

4.1 Testing the features of the DGPF for GP

To validate the utility of our framework for genetically programming sensor networks we chose an example problem well known in the area of distributed systems: the election. Election means to select one node out of a group of nodes, to act as communication relay, for instance. All nodes should receive knowledge of the id of this special node. One way to perform such an election would be to determine the maximum id of all nodes.

In order to solve this problem, we initialize all automata with their own id in the first memory cell. If an algorithm makes progress at all, the nodes should have stored greater (valid) ids there after some time. A fully functional algorithm would accomplish that the first memory cells of all nodes contain the maximum id. If the algorithm is also resource-friendly, it should reach this goal needing as few transmissions as possible.

Therefore we apply three fitness functions: the first function is the cumulative of all valid ids stored in the first memory cells of the nodes in all time steps *(i)*, see Figure 4. It is therefore an indicator both for the functionality as well as the convergence speed of the algorithms. The second fitness function is inversely proportional to the count of messages sent by all nodes *(ii)* and the third function is inversely proportional to the instruction count of the algorithms found *(iii)*.

As experimental setting we use six normal PCs in a network to perform *a)* homogeneously distributed, non-adaptive GA using the P2P-scheme described in Section 3.2 as well as *b)* randomly configured adaptive heterogeneous searches (also P2P distributed). For the experiments of type *a)*, four different population sizes are tested: 2048, 4096, 6144 and 8192. When performing the experiments of the second type, each node picks a search algorithm (GA, Hill Climbing, Simulated Annealing). If using GA it chooses a selection scheme (e.g. Tournament Selection), picks a population

size (≤ 8192), determines mutation/crossover rates, configures the caches and such and such all randomly. For each experiment, a fixed runtime of two hours is granted. The two experiments are repeated eight times each.

In Figure 4 we have plotted the fitness values of the non-dominated algorithms found by both approaches during all runs, leaving away those algorithms having minimal code size or minimal transmissions while having no functional effect at all.

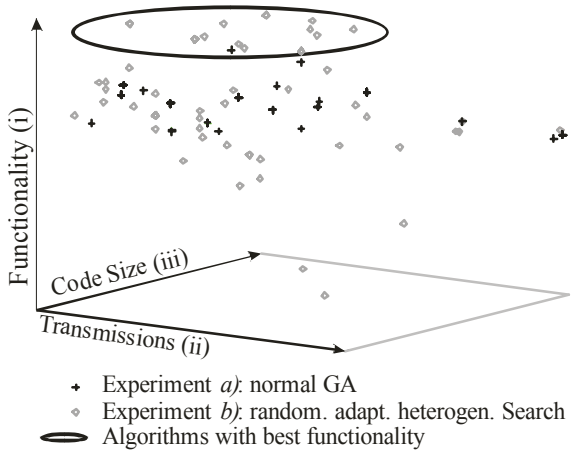

Figure 4. Utility of the DGPF-Features for GP

It now becomes clear that the auto-adaptive, randomly configured experimental setting, which takes (by chance) full advantage of all features of the DGPF, is able to find more algorithms with good functionality than a standard approach would yield. Both methods whatsoever were able to find working solutions for the election problem.

A trivial (and thus more understandable) one of these solutions is displayed in Figure 5. The algorithm consists of two parts: a procedure called when the node starts up (`procedure_0`) and an asynchronously called, interrupt-like routine which receives incoming messages (`procedure_1`). In this simple algorithm, the nodes constantly broadcast the greatest id they encountered in a loop, reducing network traffic only by performing dummy work. In Figure 4 this program is represented by a gray dot in the left of the black ellipsis.

```
called on startup                 procedure_0
store 1st variable into           0: push mem[0]
output buffer                     1: some useless operations used
                                  2: to stall and, as a consequence,
                                  3: reduce transmissions in the
                                  4: simulated/evaluated time span
send output buffer                5: send
go back to start                  6: goto 0

called asynchronously when        procedure_1
a message comes in
compare the known and             0: zf = (mem[-1] < mem[0])
the received value
if no improvement then exit       1: if zf then goto 3 // ≡exit
exchange values                   2: xchg mem[-1], mem[0]
```

Figure 5. One of the non-dominated solutions found.

5. FUTURE WORK AND CONCLUSION

There are three tasks in our research which are currently in progress and soon to be completed. The first one is the integration of additional bio-inspired search algorithms into the DGPF and the evaluation of their utility for Genetic Programming.

We will soon be able to provide an easy-to-use control and monitoring interface for the DGPF. It will graphically present the rich statistical information collected from the events created by the control FSMs. The user will be able to control a distributed search, to modify all parameters of the different nodes manually and to access the search results at any given time.

The focus of our development effort is put on Genetic Programming and its application to sensor networks. We are now able to perform research on different technologies in this area since we have laid a solid foundation of efficient search algorithms suitable for this purpose. With this foundation and the results of our future research, we hope to increase the performance of Genetic Programming and the quality of its results in that sector significantly.

In this paper we have presented a framework for heuristic randomized multi-objective search algorithms that incorporates the results of many of the best contributions to the area of randomized heuristic search. Although our own research interests concentrate on Genetic Programming, our new search API can easily be customized to any given problem space. The resulting auto-adaptive applications can be distributed over a network, performing heterogeneous, cooperative searches.

Furthermore, we provide the framework and all results to the research community under the LGPL. More information on our research as well as the fully documented Java source code of the DGPF can be found at http://dgpf.sourceforge.net [3].

6. REFERENCES

[1] Mario Villalobos-Arias, Carlos A. Coello Coello, Onésimo Hernández-Lerma: “Asymptotic Convergence of Some Metaheuristics Used for Multiobjective Optimization”, Lecture Notes in Computer Science, 2005, pages 95-111

[2] Thomas Back, Martin Schutz: “Intelligent Mutation Rate Control in Canonical Genetic Algorithms”, International Syposium on Methodologies for Intelligent Systems, 1996

[3] Distributed Genetic Programming Framework, LGPL licensed, Open-Source Java Framework, http://dgpf.sourceforge.net

[4] Thomas Weise and Kurt Geihs: "Genetic Programming Techniques for Sensor Networks", 07.2006, Proceedings of 5. GI/ITG KuVS Fachgespräch "Drahtlose

Sensornetze"

[5] Erick Cantu-Paz: “Designing Efficient Master-Slave Parallel Genetic Algorithms”, Genetic Programming 1998: Proceedings of the Third Annual Conference

[6] WS-Challenge 06 , http://insel.flp.cs.tu-berlin.de/wsc06/

[7] Fuey Sian Chong, W. B. Langdon: “Java based Distributed Genetic Programming on the Internet”, Proceedings of the Genetic and Evolutionary Computation Conference, Volume 2, Page 1229, Xear 1999

[8] W.N. Martin, J. Lienig, J. P. Cohoon (1997): “Island (migration) models: evolutionary algorithms based on punctuated equilibria”, in Back, Fogel, Michalewicz (eds.), Handbook of evolutionary Computation. IOP Publishing and Oxford University Press.

[9] Chee-Yee Chong; Kumar, S.P. Proc, "Sensor networks: Evolution, opportunities, and challenges", IEEE, August 2003

[10] “A nieched pareto genetic algorithm for multiobjective optimization”, in Proceedings of the First IEEE Conference on Evolutionary Compuation, IEEE World Congress on Computational Intelligence, vol. 1, pp 82-87, Piscataway, New Jersey, IEEE Service Centre, June 1994

[11] Astro Teller: "Turing completeness in the language of genetic programming with indexed memory", Proceedings of the 1994 IEEE World Congress on Computational Intelligence Volume 1, IEEE Press, 1994

[12] Woodward, John R.: "Evolving turing complete representations", In Congress on Evolutionary Computation, Birmingham 11th August 2003

[13] Una-May O'Reilly, Franz Oppacher: “Program Search with a Hierarchical Variable Length Representation: Genetic Programming, Simulated Annealing and Hill Climbing”, Parallel Problem Solving from Nature III, Davidor, Schwefel, Manner (Eds), Springer Verlag (LNCS), Berlin, 1994

[14] D. Levine: “Application of a hybrid genetic algorithm to airline crew scheduling”, Computers & Operations Research, 23(6):547-558, 1996

[15] H. Ishibuchi, T. Yoshida, T. Murata: “Balance between genetic search and local search in hybrid evolutionary multi-criterion optimization algorithms”, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)

[16] Leyuan Shi, Sigurdur Olafsson: “A New Hybrid Genetic Algorithm”, Late Breaking Papers at the Genetic Programming 1998 Conference, 1998, Editor: John R. Koza

[17] Vincent Bachelet, El-Ghazali Talbi: “A Parallel Co-evolutionary Metaheuristic”, Lecture Notes in Computer Science, Volume 1800, 2000

[18] Dirk Büche, Sibylle D. Müller, Petros Koumoutsakos: “Self-Adaptation for Multi-objective Evolutionary Algorithms”, Evolutionary Multi-Criterion Optimization, Proceedings of Second International Conference, EMO 2003, Faro, Portugal, April 8-11, 2003,

[19] X. Yao: “Optimization by genetic annealing”, In M. Jabri, editor, Proc. of Second Australian Conf. on Neural Networks, pp. 94-97, Sydney, Australia, 1991

[20] Luke, Sean, “ECJ: A Java-based evolutionary computation and genetic programming system”, 2000, http://cs.gmu.edu/~eclab/projects/ecj/