Using Co-solvability to Model and Exploit Synergetic Effects in Evolution

Krzysztof Krawiec and Paweł Lichocki

Institute of Computing Science, Poznan University of Technology, Poznań, Poland
krawiec@cs.put.poznan.pl, pawel.lichocki@gmail.com

Abstract. We introduce, analyze, and experimentally examine *co-solvability*, an ability of a solution to solve a pair of fitness cases (tests). Based on this concept, we devise a co-solvability fitness function that makes solutions compete for rewards granted for solving pairs of tests, in a way analogous to implicit fitness sharing. We prove that co-solvability fitness function is by definition synergistic and imposes selection pressure which is qualitatively different from that of standard fitness function or implicit fitness sharing. The results of experimental verification on eight genetic programming tasks demonstrate that evolutionary runs driven by co-solvability fitness function usually converge faster to well-performing solutions and are more likely to reach global optima.

1 Introduction

Fitness function in evolutionary algorithms is a technical means to express experimenter's expectations with respect to the final outcome of the search process. It is typically designed so as to return a maximum value for an optimal design – an ideal solution. Unfortunately, a definition of fitness function that is appropriate from experimenter's viewpoint is not necessarily also the best one for *guiding* the search in solution space.

This issue becomes more evident when one confronts the concept of fitness in evolutionary computation to its counterpart in natural evolution. In biology, fitness is an artificial gauge introduced to model the *a posteriori* probability of individual's reproduction or the changes in relative frequencies of genotypes. The particular value of such an indicator stems from innumerable interactions between the organism and its environment, including other co-evolving individuals. As such, it is inevitably a very crude derivative of individual's characteristic and cannot fully reflect the richness of all its aspects.

By an analogy to the aforementioned multiple interactions, in evolutionary computation one often simulates the behavior of a solution (its phenotype) in multiple 'environmental conditions'. This can boil down to, for instance, testing an evolved machine learning classifier on various examples, simulating an evolved robot controller in different settings, or querying an evolved function on different combinations of inputs. Each such environment, typically referred to as *fitness case* or *test*, verifies the solution on a single instance or aspect of the problem.

R. Schaefer et al. (Eds.): PPSN XI, Part II, LNCS 6239, pp. 492–501, 2010.

The outcomes of interactions with particular tests are usually additively aggregated into scalar fitness value. Analogously to natural evolution, also here such aggregation is usually simplistic and implies inevitable loss of information.

Theoreticians and practitioners of evolutionary computation have been long aware of this problem and observed its aftermaths in various undesirable phenomena, including loss of diversity and premature convergence. Diverse countermeasures has been proposed, some of which avoid aggregation into scalar fitness measure by resorting to multiple objectives, either defined explicitly by a human (evolutionary multi-objective optimization [3]), or automatically derived from problem structure (multi-objectivization [5] and underlying objectives [2]). Switching to multi-objective perspective brings however other problems, like weakened selection pressure resulting from solution incomparability (mutual non-dominance).

The method proposed in this paper relies on scalar fitness and tries to improve search convergence by adjusting the rewards assigned to solutions for coping with particular fitness cases, in a way related to implicit fitness sharing [11]. In particular, our contribution is a method that focuses on individual's ability to properly handle *pairs* of fitness cases, and treats such pairs as elementary competences (skills) for which solutions can be awarded.

2 Preliminaries

We consider here the class of iterative search problems in which solutions are evaluated on a fixed set of fitness cases. This setup is typical for, among others, genetic programming (GP), where individuals are programs (procedures) that cannot be assessed otherwise than by applying them to some external input data. This mode of evaluation can be considered as a special case of a *test-based problem*. This term has been introduced in [1] to delineate a class of coevolutionary algorithms, particularly two-population coevolution, where a population of solutions co-evolves along the population of tests. A *test* in such scenario corresponds to a fitness case in genetic programming, with the major difference being that in GP tests typically do not evolve. Because of this analogy, we will identify these notions in the remaining part of this paper and borrow some terminology from coevolutionary algorithms.

The implementation of a single act of confronting a solution s with a test t, termed *interaction* in coevolution, is problem-dependent, and can boil down to testing an evolved entity s in particular environmental conditions t, testing an evolved logical or arithmetic expression s on a specific input-output pair t, or testing a machine learning classifier s on a specific training example t. Clearly, this class of problems pertains to a great share of real-world applications.

In following, we focus on problems with binary interaction outcomes: a solution either *solves* (passes) a test or not, a fact we denote using logical predicate $s(t)$ that returns $true$ if solution s solves test t, and $false$ otherwise. Given set T of tests used to evaluate individuals in population, let $s(T) = \{t \in T : s(t)\}$ denote the subset of tests solved by s.

For this class of problems, the most straightforward way of defining fitness of a solution is to simply count the number of solved tests (a.k.a. *hits* in GP):

$$f(s) = |s(T)| \tag{1}$$

This definition, rational if no extra information on tests is available, suffers from substantial drawback: all tests contribute equally to fitness, so f can assume at most $|T|+1$ distinct values, which can result in numerous plateaus in the fitness landscape, particularly when T is small. This in turn weakens selection pressure: two solutions are likely to be indiscernible in terms of f.

A simple way of improving this state of matter is to *weigh* the rewards granted for solving particular tests. Such weights can be sometimes provided by a human expert and express his/her subjective assessment of test difficulty, test importance, or both. This, however, requires a substantial amount of domain knowledge and an extra effort. *Implicit fitness sharing* introduced by Smith *et al.* [11] and further explored for genetic programming by McKay [10,9] offers a more appealing alternative, by letting the evolution alone assess the difficulty of particular tests. Assuming that individual s is a member of population P, its fitness is here defined as:

$$f_s(s) = \sum_{t \in s(T)} \frac{1}{|P(t)|} \tag{2}$$

where $P(t) \subseteq P$ denotes the set of population members that solve test t. Thus, implicit fitness sharing simulates limits imposed on resources: individuals *share* the rewards for solving particular tests, each of which can vary from $\frac{1}{|P|}$ to 1 inclusive. Higher rewards are provided for solving tests that are rarely solved by population members (small $P(t)$), while importance of tests that are easy (large $P(t)$) is diminished. Additionally, because $P(t)$ typically pertains to the current population only, the assessed difficulties of tests change with time, which can help the search process escape local minima (as opposed to fixed weighting).

As such, fitness sharing can be perceived as a simple form of coevolution, where individuals compete for tests and their fate depends on the performance of other individuals (though there are no direct, face-to-face interactions between individuals). From yet another perspective, fitness sharing is a diversity maintenance technique: an individual that solves a low number of tests can still survive if its competence is rare. In this way, implicit fitness sharing helps reducing crowding and premature convergence; it shares this objective with explicit fitness sharing proposed in [4], where population diversity is enforced by monitoring genotypic or phenotypic distances between individuals.

3 Co-solvability

In broader terms, implicit fitness sharing enables an evolutionary process to assess the relative importance of *skills*, where skill is identified with the ability to solve a particular test. In real world however, it is often the *combination* of skills that matters. For an animal, the skill of digging and the skill of navigation

can bring substantial benefits independently. However, when combined, they enable finding the previously buried prey and survive when food is scarce, a benefit which can be greater than the sum of benefits of its constituents. As another example, the overall performance of a mobile robot that is intended to move around a building depends on multiple skills, like the ability to move straight, the ability to make precise turns, and the ability to estimate its position. Again, each of these skills alone is not enough for the completion of the task, but together they make it possible.

Implicit fitness sharing cannot model such nonlinear accumulation of skills: the reward for simultaneous mastering of two or more skills amounts to the sum of rewards obtained for each skill individually. To enable synergy between pairs of skills, we introduce the notion of *co-solvability*. We call a pair of tests (t_i, t_j) *co-solvable by* s if and only if $s(t_i) \wedge s(t_j)$. The *co-solvability matrix* for a population P evaluated on set of tests T is a $|T| \times |T|$ matrix, with elements defined as

$$c_{ij} = \begin{cases} |\{s \in P : s(t_i) \wedge s(t_j)\}|, & i \leq j \\ 0, & otherwise \end{cases}$$

This matrix allows us to define the *co-solvability fitness function* f_c that rewards individuals for solving pairs of *distinct* tests:

$$f_c(s) = \sum_{t_i, t_j \in T : s(t_i) \wedge s(t_j), i<j} \frac{1}{c_{ij}} \tag{3}$$

Similarity of this formula to Formula (2) is not coincidental: co-solvability can be viewed as second-order fitness sharing. Let us notice that sharing of rewards for co-solving particular pairs of tests is an essential component here: simply *counting* the co-solvable tests ($|\{(t_i, t_j) : s(t_i) \wedge s(t_j),\ i < j\}|$) orders solutions in the same way as the standard fitness measure (Formula (1)), yielding precisely the same proceeding of evolution under any rank-based selection (e.g., tournament selection).

4 Properties of Co-solvability

Let us start from noticing that co-solvability fitness function f_c, similarly to f and f_s, fulfills the fundamental property we expect from any test-based fitness function, i.e., it is monotonous with respect to inclusion of sets of tests solved: for any $s_1, s_2, s_1(T) \subset s_2(T)$, it holds that $f_c(s_1) < f_c(s_2)$.

Let us consider four solutions s_1, s_2, s_3, s_4 such that when tested on four tests t_1, t_2, t_3, t_4, they perform as shown in Table 1a. Next, let us assume that the population contains a copies of s_1, b copies of s_2, c copies of s_3, and d copies of s_4[1]. The co-solvability matrix C for this population is shown in Table 1b.

[1] In other words, we work here with equivalence classes of solutions rather than with single solutions.

Table 1. An exemplary problem: the performances of solutions on tests (a) and the corresponding co-solvability matrix (b). Empty cells denote zeroes.

(a)

	t_1	t_2	t_3	t_4
s_1	1	1	0	0
s_2	0	0	1	1
s_3	0	1	1	0
s_4	1	0	0	1

(b)

	t_1	t_2	t_3	t_4
t_1	a+d	a		d
t_2		a+c	c	
t_3			b+c	b
t_4				b+d

Table 2 presents the fitness values for solutions $s_1 \ldots s_4$ as assigned by particular fitness functions: standard fitness f (Eq. (1)), fitness sharing f_s (Eq. (2)), and co-solvability fitness function f_c (Eq. (3)). We note that f does not discern any pair of solutions, no matter how often they occur in population. The ability of f_s and f_c to discern solutions depends on the actual values of a, b, c and d.

Let us use the solutions s_1 and s_3 from the above example to demonstrate that f_c can produce different ordering of individuals than fitness sharing. Technically, we want to check whether it is possible for $f_s(s_1) < f_s(s_3)$ and $f_c(s_1) > f_c(s_3)$ to hold simultaneously. As it follows from Table 2, these two conditions are respectively equivalent to $a+d > c+b$ and $a < c$, which are fulfilled by infinitely many quadruples of $a, b, c, d \geq 0$. Therefore, f_c is able to order solutions differently from f_s. Quite interestingly, it can be proven that four is the minimal number of tests required to produce such difference.

Let us now translate this observation into evolutionary context and give examples of scenarios when f_c produces substantially different results than f_s:

1. Consider two individuals s_1, s_2 such that $s_1(T) \cap s_2(T) = \emptyset$. Assume they undergo crossover and produce offspring s such that $s(T) = s_1(T) \cup s_2(T)$. Under f_c it has to hold $f_c(s) > f_c(s_1) + f_c(s_2)$, whereas for f and f_s equalities would hold. Thus, co-solvability is not additive and *enforces* synergy: for parents that exhibit mutually exlusive skills, their offspring that adopts all their skills is by definition better than both of them taken together. Also, f_c can be considered non-Markovian with respect to the changes observed in $s(T)$ as s undergoes evolutionary modifications: the increase of individual's fitness resulting from acquiring an ability of solving another test depends on the set of tests already solved by that individual.

2. Consider two individuals s_1, s_2 such that $s_1(T) \neq s_2(T)$ and $f_c(s_1) > f_c(s_2)$, and a test t such that $t \notin s_1(T) \cup s_2(T)$. Then, let us assume that, as a result of genetic modification both s_1 and s_2 acquire the skill of solving t, so that for the resulting solutions s'_1 and s'_2 it holds $s'_1(T) = s_1(T) \cup \{t\}$ and $s'_2(T) = s_2(T) \cup \{t\}$. As a conclusion from the above analysis, it is possible that $f_c(s'_1) < f_c(s'_2)$. In other words, s_2 can gain more from the same modification than s_1. Neither standard fitness nor implicit fitness sharing allow such possibility (the offspring of s_1 would be better than the offspring of s_2).

Table 2. Fitness values assigned to individuals from Table 1 by particular fitness functions

Fitness definition	s_1	s_2	s_3	s_4
Standard fitness f	2	2	2	2
Implicit fitness sharing f_s	$\frac{1}{a+d}+\frac{1}{a+c}$	$\frac{1}{b+c}+\frac{1}{b+d}$	$\frac{1}{a+c}+\frac{1}{b+c}$	$\frac{1}{a+d}+\frac{1}{b+d}$
Co-solvability fitness f_c	$\frac{1}{a}$	$\frac{1}{b}$	$\frac{1}{c}$	$\frac{1}{d}$

Co-solvability fitness is usually less discrete than f and f_s. For $n = |T|$ tests, standard fitness function f can return only $n+1$ distinct values. For the fitness sharing method, that number amounts to $n^{|P|}$ if the population is sufficiently large (precisely: if $|P|$ is greater than the n^{th} prime number[2]). For co-solvability, $[\frac{n(n-1)}{2}]^{|P|}$ distinct values are possible.

In [7], Lasarczyk *et al.* proposed a method for selection of fitness cases based on a concept similar to co-solvability. The method maintains a weighted graph that spans fitness cases, where the weight of an edge reflects the historical frequency of a pair of tests being solved simultaneously. Fitness cases are selected based on a sophisticated analysis of that graph. Compared to that, our co-solvability is a simpler, parameter-free approach, which does not *select* the fitness cases but *weighs* pairs of them, individually for each solution (in [7], the same selected subset of fitness cases is used for all solutions).

5 The Experiment

The above analysis proves that co-solvability measure can produce different orderings of solutions and thus potentially steer evolution in other directions than conventional fitness measure f and fitness sharing f_s. It is however far from obvious whether this change is beneficial for effectiveness of search. In this section, we verify whether the fitness pressure imposed by co-solvability improves the performance of an evolutionary run applied to typical problems of logical function synthesis: multiplexer, parity, and two types of comparators. To this aim, we apply the approach of genetic programming (GP), with individuals being programs (procedures) encoded as expression trees.

For each problem, we prepared two instances, small and large, differing in the number of inputs. Table 3 summarizes the four considered problems, listing for each problem instance the number of inputs (independent variables, bits), the number of tests ($|T|$), and the proportion of tests for which the output of the program should be 1 and such for which the output should be 0. In the *Parity-odd* problem, the task is to evolve an expression that returns true if an odd number of ones appears on its inputs. *Multiplexer* should return the same value as the state of the addressed input (6-bit multiplexer uses two inputs to

[2] Sketch of proof: if $|P(t)|$ is a distinct prime number for each t, every combination of rewards received for particular tests yields a unique value of fitness f_s (Formula 2).

Table 3. The summary of problems and problem instances

	Small instance			*Large instance*		
Problem	*Inputs*	*Tests*	*Proportions*	*Inputs*	*Tests*	*Proportions*
Parity-odd	5	32	16:16	6	64	32:32
Multiplexer	6	64	32:32	11	2048	1024:1024
Eq	6	64	8:56	8	256	16:240
Cmp	6	64	28:36	8	256	120:136

Table 4. Success rates of best-of-run individuals produced by the methods, defined as the probability of run producing an ideal solution estimated from the total of 30 runs

	Small instance			*Large instance*		
Problem	f	f_s	f_c	f	f_s	f_c
Parity-odd	0.133	0.800	0.967	0.000	0.000	0.067
Multiplexer	1.000	1.000	1.000	0.433	0.700	0.567
Eq	0.000	1.000	1.000	0.000	0.700	0.933
Cmp	0.633	1.000	1.000	0.133	0.800	0.933

address the remaining four inputs, 11-bit multiplexer uses three inputs to address the remaining eight inputs). m-bit comparator *Eq* returns one if the $\frac{m}{2}$ least significant input bits encode a number that is equal to the number represented by the $\frac{m}{2}$ most significant bits. The *Cmp* comparator does the same but only when the former of these numbers is smaller than the latter.

Concerning GP-specifics, we use the Koza-I-style setup [6] with some modifications. The most important settings include: population of 1024 individuals initialized using the standard ramped half-and-half method, tournament selection with tournament of size 7, tree-swap crossover engaged with probability 0.9, subtree-replacing mutation applied with probability 0.1, no elitism. Evolution lasts for 200 generations. The software testbed has been implemented with help of ECJ [8] and is available at http://www.cs.put.poznan.pl/kkrawiec.

Table 5. Success effort (the expected number of generations to find the ideal)

	Small instance			*Large instance*		
Problem	f	f_s	f_c	f	f_s	f_c
Parity-odd	1448	159	102	∞	∞	2999
Multiplexer	9	8	8	370	164	224
Eq	∞	38	33	∞	192	121
Cmp	186	32	31	1453	181	136

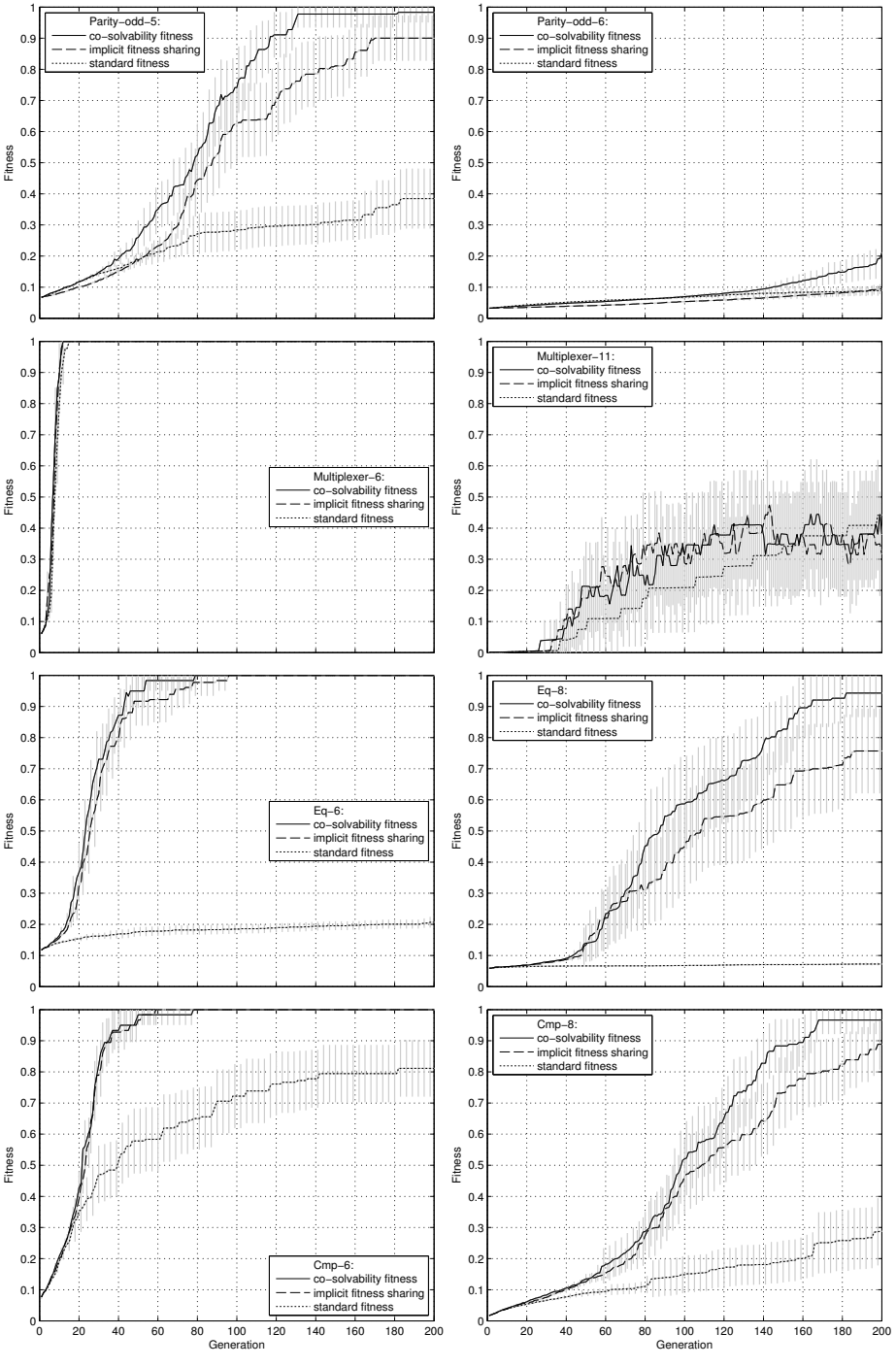

Fig. 1. Best-of-generation fitness with ±.95 confidence, averaged over 30 runs

Table 4 reports the success rates of the best-of-run individuals produced by the runs that used standard fitness f, implicit fitness sharing f_s, and co-solvability fitness f_c (because fitness definition is the only difference between the setups, in following we refer to them using these symbols). We define success rate as the estimated probability of run producing an ideal solution estimated from the total of 30 runs.

The results confirm the common wisdom that *Parity* problems are the hardest in the considered group. The function to be learned here exhibits highest possible input-output sensitivity: the output flips every time a single input changes state. For the small 5-bit instance, this difficulty seems to be tractable, but the large instance (only one more input) renders the problem hopelessly difficult for f and f_s. However, we note that co-solvability still manages to find an ideal solution twice per 30 runs, which is not much, but qualitatively better than f and f_s.

It also turns out that, except for *Multiplexer-11*, f_c significantly outperforms the other approaches on the remaining instances. This happens whenever there is some space for improvement; otherwise, it does not yield to f_s. The gains in performance appear more convincing when expressed in terms of success effort reported in Table 5, which we define as the sum of generations in which an ideal was found divided by the number of successful runs (this is a pessimistic estimate of the expected number of generations required to find an ideal solution, which yields ∞ if none of 30 runs succeeds).

Though the failure of f_c on the *Multiplexer-11* problem requires deeper investigation, we hypothesize that it is the number of tests that is here the culprit: 2048 tests means over four million elements in the co-solvability matrix.

Figure 1 presents the fitness graphs for all problems averaged over 30 runs. Because the values of functions that propel evolution in particular methods (f, f_s, and f_c) are mutually incomparable, we plot here fitness defined in the same way for all approaches, i.e. as $f'(s) = 1/(1 + |T| - |s(T)|)$, which returns a small positive number for the worst possible individual ($s(T) = \emptyset$), and 1.0 for an ideal ($s(T) = T$). Because of nonlinear definition of f', the differences observed in plots mean rather moderate gains in terms of the number of tests solved. However, the superiority of f_c should be judged substantial, as in the domain of logical function synthesis any deviation from the ideal solution essentially renders the solution useless.

Most importantly, f_c turns out to be never significantly worse than f_s according to Wilcoxon runk-sum test for equal medians of fitness f' applied to 200^{th} generation. And, for both *Parity* instances and both *Eq* instances, f_c is significantly better ($p < 0.05$).

6 Conclusion

We demonstrated here that fitness function based on a simple concept of co-solvability is qualitatively different from the standard definition of fitness and implicit fitness sharing and brings in substantial benefits for an evolutionary search. Though the experimental evaluation comprised only problems from the

realm of genetic programming, the proposed method abstracts from representation of solutions and is thus applicable to any test-based search problem. It is likely then that similar gains could be observed for other classes of problems.

The method is straightforward to implement and its computational overhead is basically the same as in case of fitness sharing. This extra cost can be considered negligible when compared to the actual cost of testing a solution on a test (i.e., determining the outcome of $s(t)$), which is typically much higher.

Acknowledgment. This work was supported in part by Ministry of Science and Higher Education grant # N N519 3505 33.

References

1. Bucci, A., Pollack, J.B., de Jong, E.: Automated extraction of problem structure. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 501–512. Springer, Heidelberg (2004)
2. de Jong, E.D., Pollack, J.B.: Ideal Evaluation from Coevolution. Evolutionary Computation 12(2), 159–192 (Summer 2004)
3. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester (2001)
4. Goldberg, D.: Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading (1989)
5. Knowles, J.D., Watson, R.A., Corne, D.: Reducing local optima in single-objective problems by multi-objectivization. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 269–283. Springer, Heidelberg (2001)
6. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
7. Lasarczyk, C.W.G., Dittrich, P., Banzhaf, W.: Dynamic subset selection based on a fitness case topology. Evolutionary Computation 12(2), 223–242 (Summer 2004)
8. Luke, S.: ECJ evolutionary computation system (2002), `http://cs.gmu.edu/eclab/projects/ecj/`
9. McKay, R.I.B.: Committee learning of partial functions in fitness-shared genetic programming. In: 26th Annual Confjerence of the IEEE Third Asia-Pacific Conference on Simulated Evolution and Learning 2000, Industrial Electronics Society, IECON 2000, Nagoya, Japan, October 22-28, vol. 4, pp. 2861–2866. IEEE Press, Los Alamitos (2000)
10. McKay, R.I.B.: Fitness sharing in genetic programming. In: Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.-G. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000), Las Vegas, Nevada, USA, July 10-12, pp. 435–442. Morgan Kaufmann, San Francisco (2000)
11. Smith, R., Forrest, S., Perelson, A.: Searching for diverse, cooperative populations with genetic algorithms. Evolutionary Computation 1(2) (1993)