4

Behavioral assessment of test difficulty

As argued in Sect. 2.2.2, one of the vices of conventional scalar evaluation is symmetry: the same reward is granted for passing every test. Yet some tests can be objectively more difficult than others in the sense of (2.2), i.e. harder to pass by a randomly generated program. They may vary also with respect to subjective difficulty, i.e. particular program synthesis methods may find it more or less difficult to synthesize a program that passes a given test (cf. Sect. 2.2.3). Conventional evaluation function (1.7), by simply counting the failed tests, cannot address this aspect of program synthesis.

In theory, test difficulty can be obtained from domain knowledge or provided by a human expert. But human expertise and domain knowledge are not always available or affordable, not to mention the extra effort required in such scenarios.

In this chapter, we show how information on test difficulty can be conveniently acquired from an execution record and used to redefine an evaluation function. This idea materialized originally in GP with the advent of *implicit fitness sharing* [166], which we cover in Sect. 4.2. In subsequent sections, we present the conceptual progeny of that approach: the methods that scrutinize cosolvability of tests [94] (Sect. 4.3) and automatically derive objectives from interaction matrices [112, 95] (Sect. 4.4). Before presenting that material, we first introduce the test-based perspective on program synthesis, which comes in particularly handy for this kind of considerations.

4.1 Test-based problems

Evaluation in GP can be alternatively phrased as candidate programs engaging in *interactions* with tests. In that framing, the evaluation of a candidate program p depends on an *interaction function* $g : \mathcal{P} \times \mathcal{T} \to \{0, 1\}$, which is an indicator function of the set of tests passed by p, i.e.

interaction function

$$g(p, t) = g(p, (in, out)) = [p(in) = out], \tag{4.1}$$

K. Krawiec, *Behavioral Program Synthesis with Genetic Programming*,
Studies in Computational Intelligence 618,
DOI: 10.1007/978-3-319-27565-9_4

where [ ] is the Iverson bracket (2.4). The objective function f_o (1.7) can be then rewritten as

$$f_o(p) = \sum_{(in,out)\in T} g(p,(in,out)). \tag{4.2}$$

For convenience, we will occasionally abuse notation and treat g as a logical predicate, i.e. write $g(p,t)$ when $g(p,t)=1$ and $\neg g(p,t)$ when $g(p,t)=0$.

interaction matrix

The outcomes of interactions of all programs in a given population P with all tests from a given set T can be gathered in an *interaction matrix*

$$G = [g_{ij} = g(p_i,t_j) : p_i \in P, t_j \in T]. \tag{4.3}$$

Note that the ith row of G is the outcome vector for p_i, i.e. $o_T(p_i)$ (2.3), or in other words the rightmost column of an execution record (Fig. 3.2).

coevolutionary algorithm

test-based problem

Formalizing an evaluation function in terms of interactions is not particularly common in GP and more typical for coevolutionary algorithms (CoEAs, [149]), where it originated in *test-based problem* [16, 24]. In a test-based problem, one seeks an element or a combination of elements from solution space $\mathcal{S}$ that conforms a given *solution concept* [30, 149]. The arguably simplest example of solution concept is *maximization of expected utility*, i.e. a candidate solution that maximizes the expected interaction outcome, i.e. $\arg\max_{s\in\mathcal{S}} \mathbb{E}_{t\in\mathcal{T}} g(s,t)$.

The number of tests in $\mathcal{T}$ is usually large or infinite, which makes it technically infeasible to elicit exact values of an evaluation function. This problem can be addressed by sampling the tests to be used for evaluation, which can be done in at least three ways. In the simplest scenario, a sample T is drawn once from $\mathcal{T}$ and remains fixed throughout a run of a method; this case resembles a typical GP setup the most. Alternatively, one may sample T from $\mathcal{T}$ repetitively, for instance in each generation [21]. The third way is to let the tests *coevolve* with the candidate solutions in a CoEA framework. Typically, candidate solutions and tests co-evolve in two separate populations $S \subset \mathcal{S}$ and $T \subset \mathcal{T}$, respectively, interacting with each other only for the purpose of evaluation. While the candidate solutions are, as usually, rewarded for *performing*, the tests are rewarded for *informing*, for instance for the number of pass-fail distinctions they make between the current candidate solutions. The underlying rationale is that a CoEA can autonomously induce a useful search gradient by assorting the tests, and find good solutions faster or more reliably and/or at a lower computational cost compared to using all tests from $\mathcal{T}$ (where feasible) or drawing them at random. Empirical evidence gathered in previous work on various test-based problems suggests that this is indeed possible, provided proper tuning of a CoEA [69, 175].

Though CoEAs are not explicitly used in the methods studied in this book, the test-based framework is convenient for capturing the diversity of behaviors in an evolving population. Crucially, it allows to juxtapose not only the behaviors of programs, but also compare the characteristics of tests, which is the idea behind the approaches presented in next sections.

4.2 Implicit fitness sharing

implicit fitness sharing

Implicit fitness sharing (IFS) introduced by Smith et al. [166] and further explored in GP by McKay [124, 123] originates in the observation that difficulty of particular tests may vary. Let us reiterate after Sect. 2.2.2 that problems with uniform distribution of test difficulty are less common than problems where difficulty varies by tests, as the former is a special case of the latter. The conventional objective function (1.7) is oblivious to that fact and grants the same reward of 1 for solving every test in T, which may result in premature convergence discussed in Sect. 2.4. In order to entice a search process to pass the more difficult tests, one might want to increase the rewards for them. But where to look for reliable information on test difficulty? The exact objective difficulty (2.2) and subjective difficulty introduced in Sect. 2.2.2 are of little use here: the former requires running all programs in $\mathcal{P}$ on a given test, and the latter estimating the probability that a given synthesis algorithm produces a program that passes a given test.

To estimate the difficulty of particular tests in T, IFS uses the outcomes of their interactions with the candidate programs in the working population $P \subset \mathcal{P}$, and defines the evaluation function as follows:

$$f_{\text{IFS}}(p) = \sum_{t \in T : g(p,t)} \frac{1}{|P(t)|} \tag{4.4}$$

where $P(t) \subseteq P$ denotes the subset of population members that pass test t:

$$P(t) = \{p \in P : g(p,t)\}. \tag{4.5}$$

Notice that $P(t)$ corresponds to a column in an interaction matrix (4.3), and $|P(t)|$ is equal to a sum of such a column.

In contrast to evaluation functions considered so far, f_{IFS} is *maximized*. The denominator in Formula 4.4 never becomes zero, because if p passes a given t, then $P(t)$ must contain at least p. The computational overhead of calculating f_{IFS} is usually negligible, because to get evaluated, the programs in P have to be applied to the tests in T anyway.

Example 4.1. Consider a population of three programs $P = \{p_1, p_2, p_3\}$ evaluated on four tests $T = \{t_1, t_2, t_3, t_4\}$, with interaction matrix shown in the left part of Table 4.1. Although p_1 and p_2 pass the same number of tests, p_1 is granted greater value of f_{IFS} because it passes the tests that no other program in P passes. On the other hand, p_2 is not unique in P in its capability of passing t_3. Thus, $f_{\text{IFS}}(p_1) > f_{\text{IFS}}(p_2)$, even though $f_o(p_1) = f_o(p_2)$. ■

Table 4.1: Calculation of IFS evaluation for an exemplary population P and four tests in T. The upper left 3×4 part of the table presents the matrix G of interaction outcomes between P and T. The bottom row shows the number of programs in P that pass a given test. The column marked $f_o(p_i)$ presents the conventional objective function, i.e. the number of failed tests. The rightmost column shows the calculation of IFS evaluation, which results from sharing the rewards for solving particular tests. Note that an individual's evaluation is simply the scalar product of its outcome vector with the vector of inverted cardinalities of $P(t)$s.

G	t_1	t_2	t_3	t_4	$f_o(p_i)$	$f_{\text{IFS}}(p_i)$
p_1	1	1	0	0	2	$1+1+0+0=2$
p_2	0	0	1	1	2	$0+0+\frac{1}{2}+1=\frac{3}{2}$
p_3	0	0	1	0	3	$0+0+1+0=1$
$\lvert P_{t_i}\rvert$	1	1	2	1		

The key characteristics of IFS is that it estimates difficulty from an evolved population of programs, i.e. a sample that is biased by a specific selection pressure. The term $\frac{1}{|P(t)|}$ in (4.4) is IFS's measure of difficulty of test t, which depends reciprocally, and thus non-linearly, on the number of programs that pass t (contrary to objective test difficulty (2.2)). As a consequence, tests in IFS can be likened to limited resources: individuals in a population *share* the rewards for solving them, where a reward can vary from $\frac{1}{|P|}$ to 1 inclusive. Higher rewards are granted for tests that are rarely passed by population members (small $P(t)$), and lower for the tests passed frequently (large $P(t)$). Allocation of rewards depends on the capabilities of the current population and is in this sense *relative* rather than objective or subjective. Despite this transient nature, empirical evidence shows that f_{IFS} can substantially improve performance compared to the conventional objective function f_o [114, 98].

relative test difficulty

The relative nature of f_{IFS} makes it different from conventional evolutionary algorithms, where an evaluation of a candidate solution is normally *context-free*, i.e. does not depend on the other candidate solutions. IFS may thus seem to resemble a coevolutionary algorithm (Sect. 4.1). However, in coevolutionary algorithms, individuals interact with each other directly, while in IFS there is no face-to-face competition between them. Interestingly, IFS can be also remotely related to *shaping*, an extension of the conventional reinforcement learning paradigm [173]: by varying the rewards for solving particular tests, IFS can be said to modify its own *training experience* [175].

Because IFS increases the survival odds for candidate solutions that have 'rare competences', it is commonly considered as a diversity maintenance technique and a means of avoiding *premature convergence*. These characteristics motivated also *explicit fitness sharing* proposed in [41], where population diversity is encouraged by monitoring genotypic or phenotypic distances between individuals. By allowing the same program to receive

diversity maintenance

explicit fitness sharing

different evaluation in particular generations of an evolutionary run, IFS may also facilitate escaping from local minima.

IFS assumes interaction outcomes to be binary: the tests that have been passed by a program need to be clearly delineated from those that have not. In real-valued domains, that concept is in a sense ‘fuzzified’ and programs can perform better or worse on individual tests. In [98] we proposed a generalized variant of IFS that ranks programs in a population with respect to errors they commit on a given test and obtain so reliable information on test difficulty. The method achieved top accuracy when confronted with several other extensions of GP on a nontrivial real-world task of detection of blood vessels in retinal imagining.

4.3 Promoting combinations of skills via cosolvability

A program’s capability in passing a given test can be likened to a *skill*. We presented that perspective in Sect. 2.3. It reverberates with some earlier works, albeit within very different formal frameworks (for instance production rules, conditional programs, and analogical problem solving in [159]). IFS defines evaluation as a sum of rewards for mastering individual skills. In real world however, it is often the *combination* of skills that matters. Reaching for a biological analogy, the skill of digging in the ground and the skill of navigation may each on its own bring only marginal benefits for an animal. However, when combined, they enable finding previously-buried prey and hence survival when food is scarce, an advantage which can be greater than the sum of the constituent benefits. As another example, the overall performance of a mobile robot may depend on multiple skills, including the ability to maintain a straight-line trajectory, the ability to turn, and the ability of position estimation. Each of these skills alone may be insufficient to complete a given task, but together they may make that possible.

skill

In IFS, the reward for passing two tests simultaneously amounts to the sum of rewards obtained for passing each test individually (4.4). IFS cannot thus model *synergy*, i.e. reward a combination of skills higher than the sum of rewards of its constituents. To model non-additive interactions between skills, in [94] we introduced the notion of *cosolvability*. We call a pair of tests (t_i, t_j) *cosolvable by* a program p if and only if p passes both of them, i.e. $g(p, t_i) \wedge g(p, t_j)$. The *cosolvability matrix* for a population P evaluated on tests in T is a symmetric $|T| \times |T|$ matrix C, with the elements defined as

synergy of skills

cosolvability

cosolvable tests

$$c_{ij} = |\{p \in P : \, g(p, t_i) \wedge g(p, t_j)\}|, \tag{4.6}$$

We define then the *cosolvability evaluation function* f_{CS} that rewards programs for solving pairs of *distinct* tests:

Table 4.2: An interaction matrix G for an exemplary population of four programs and for four tests (a), and the corresponding cosolvability matrix C (b). Empty cells denote zeroes.

(a)

G	t_1	t_2	t_3	t_4
p_1	1	1	0	0
p_2	0	0	1	1
p_3	0	1	1	0
p_4	1	0	0	1

(b)

C	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 4.3: Fitness values assigned to programs from Table 4.2a by particular evaluation functions.

Evaluation function	p_1	p_2	p_3	p_4
f_o	2	2	2	2
f_{IFS}	$\frac{1}{a+d} + \frac{1}{a+c}$	$\frac{1}{b+c} + \frac{1}{b+d}$	$\frac{1}{a+c} + \frac{1}{b+c}$	$\frac{1}{a+d} + \frac{1}{b+d}$
f_{CS}	$\frac{1}{a}$	$\frac{1}{b}$	$\frac{1}{c}$	$\frac{1}{d}$

$$f_{\text{CS}}(p) = \sum_{i<j, c_{ij}>0} \frac{1}{c_{ij}} \tag{4.7}$$

As f_{IFS}, f_c is *maximized.* Similarity of this formula to (4.4) is not incidental: cosolvability can be viewed as a form of second-order fitness sharing, it is the rewards for solving *pairs* of tests that is shared.

Example 4.2. Consider four programs p_1, p_2, p_3, p_4 that perform on tests t_1, t_2, t_3, t_4 as shown in the interaction matrix in Table 4.2a. Assume that the population P contains a programs that produce the same outcome vector as p_1, i.e. a 'behavioral clones' of p_1. Similarly assume b behavioral copies of p_2, c copies of p_3, and d copies of p_4. The cosolvability matrix C for this population is shown in Table 4.2b. Note that co-occurrence of multiple programs that have the same outcome vector is likely in a population of programs that have been evolving for some time.

Table 4.3 presents the evaluations for programs $p_1 \dots p_4$ as assigned by particular evaluation functions: conventional f_o (1.7), fitness sharing f_{IFS} (4.4), and cosolvability evaluation function f_{CS} (4.7). We note that f_o does not discern programs at all, no matter how often they occur in the population. Whether f_{IFS} and f_{CS} discern particular pairs of programs depends on the numbers of occurrences of $t_1, \dots, t_4$, i.e. on of a, b, c and d.

Programs p_1 and p_3 allow us to demonstrate that f_{CS} can produce different ordering of individuals than fitness sharing. Let us see if $f_{\text{IFS}}(p_1) < f_{\text{IFS}}(p_3)$ and $f_{\text{CS}}(p_1) > f_{\text{CS}}(p_3)$ can hold simultaneously. As it follows from Table 4.2, 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_{CS} can order solutions differently from f_{IFS} and, in consequence, lead to different outcomes of a GP run. ■

Let us now investigate the ability of f_{IFS} to model synergy between skills. Let $T(p)$ denote the set of tests from T that are passed by p, i.e,

$$T(p) = \{t \in T : g(p, t)\}. \tag{4.8}$$

$T(p)$ corresponds to a row of an interaction matrix (4.3), in analogy to $P(t)$ (4.5) that corresponds to a column.

Consider two programs p, p' such that $T(p) \cap T(p') = \emptyset$. Assume they are crossed over and produce an offspring p_o that is their perfect 'behavioral mixture', i.e. inherits all skills from them and does not have any other skills. Formally,

$$T(p_o) = T(p) \cup T(p'). \tag{4.9}$$

It obviously holds that $f_o(p_o) = f_o(p) + f_o(p')$, because f_o simply counts the passed tests. f_{IFS} is similarly additive and an analogous relation $f_{\text{IFS}}(p_o) = f_{\text{IFS}}(p) + f_{\text{IFS}}(p')$ holds, though for this to be true we need to assume that p_o, p, and p' are members of the same population for which the subjective difficulty of tests is estimated. For the sake of argument, we will stick to this assumption this for the rest of this section.

In contrast to f_o and f_{IFS}, CS is not additive in the above sense. The offspring not only inherits the scores earned by its parents, but also receives additional rewards for passing the pairs of tests the parents have individually failed. In effect, it is guaranteed that $f_{\text{CS}}(p_o) > f_{\text{CS}}(p) + f_{\text{CS}}(p')$. Thus, cosolvability not only enables, but actually *enforces* synergy: an offspring that inherits all skills from parents that have mutually exclusive skills is by definition better than both of them taken together. Given the relative nature of cosolvability, the actual differences in evaluation vary depending on the skills of other programs in a population, nevertheless the above statement is guaranteed to hold.

Now consider two programs p, p' such that $T(p) \neq T(p')$ and $f_{\text{CS}}(p) > f_{\text{CS}}(p')$, and a test t such that $t \notin T(p) \cup T(p')$. Assume that, as a result of modification, both p and p' acquire the skill of passing t, so that for the respective resulting offspring programs o and o' it holds $T(o) = T(p) \cup \{t\}$ and $T(o') = T(p') \cup \{t\}$. From the above analysis it follows that $f_{\text{CS}}(o) < f_{\text{CS}}(o')$ is possible. Thus, o' can gain more than o for passing the same test, to the extent that it becomes better than o, even though its parent was worse than the parent of o. Neither the conventional objective function f_o nor IFS allow for that; under both these evaluation functions, o is better than o'.[1]

[1] These observations hold for ordinal selection methods that care only about the ordering of solutions (e.g., tournament selection). For selection methods that

The above properties cause the dynamics of an evolutionary search under f_{CS} to be in general different from that of f_{IFS} and f_o. The differences stem not only from the co-occurrence of skills in a population, but also from the sizes of P and T which determine the likelihood of ties on evaluation. As we noticed in Sect. 2.1, f_o can return only $|T|+1$ distinct values, so if $|P| \gg |T|$, ties on f_o become likely. For IFS and cosolvability, a complementary relationship holds: the greater the number of programs in P, there more likely it is that different tests are passed by different numbers of programs and, as a consequence, programs are granted different evaluations. In general, ties are thus less likely for IFS than for the conventional evaluation function, and even less likely for cosolvability.

The synergistic nature and fine-grained codomain of f_{CS} proved beneficial in the empirical examination we reported in [94], where, with exception of one out of eight benchmarks, it improved the likelihood of successful program synthesis in comparison to f_o and f_{IFS}. We are aware of only one technical inconvenience of this approach: the size of cosolvability matrix is quadratic with respect to the number of tests, so its memory occupancy may become noticeable when the number of tests reaches the order of thousands.

A concept vaguely related to cosolvability was subject of the study by Lasarczyk et al. [109]. The authors proposed there a method of test selection that maintains a weighted graph that spans tests, where the weight of an edge reflects the historical frequency of a pair of tests being passed simultaneously. The graph is analyzed to select the 'essential' tests that are then used to evaluate all individuals in population. Compared to that approach, cosolvability is a simpler, parameter-free approach, which does not *select* the tests but *weighs* pairs of them, and does that individually for each evaluated program.

4.4 Deriving objectives from program-test interactions

Pareto coevolution

elementary objective

The concept of interaction matrix (4.3) naturally leads to the idea of *Pareto coevolution* [31, 137], where aggregation of interaction outcomes is abandoned in favor of treating each test as an *elementary objective* and comparing candidate solutions with *dominance relation*, as we did in Sect. 2.2.3 with lattices of outcome vectors (Fig. 2.1). A candidate solution p_1 dominates p_2 if and only if it performs at least as good as p_2 on all tests, and strictly better on at least one test. For instance, p_2 in Table 4.1 dominates p_3 as it passes all the tests passed by p_3 and t_4, which p_3 does

assume evaluation to be defined on a metric scale (like fitness-proportionate selection), it becomes even easier for f_{CS} to produce evaluations that imply different selection probabilities than those of f_{IFS}.

not pass. On the other hand, there is no dominance between p_1 and p_2 – none of these programs is clearly better than the other.

dominance relation on tests

In principle, dominance relation on tests (*dominance on tests* in the following) can be directly used to determine the outcomes of selection in an evolutionary loop of GP [88]. The arguably simplest selection operator of this kind would, given a pair of programs, return the one that dominates the other, or pick any of them at random in case of mutual non-dominance. However, when the number of tests is large, dominance between candidate solutions becomes unlikely, as there is high chance that each of compared solutions passes a test that the other solution fails. The dominance relation becomes sparse, with many pairs of candidate solutions left incomparable. This in turn weakens the search gradient, and makes search process less effective.

coordinate systems

underlying objective

The limitations of dominance on tests as a means for selection of candidate solutions sparked search for alternative means of exploiting interaction matrices. The breakthrough came with the observation that test-based problems may feature an *internal structure*. Bucci [16] and de Jong [23] introduced *coordinate systems* that *compress* the elementary objectives (each associated with a unique test) into a multidimensional structure of *underlying objectives* (dimensions), while preserving the dominance relation between candidate solutions. Because some tests can be redundant, the number of underlying objectives can be lower (and, interestingly, may indicate the inherent complexity of a given test-based problem).

However, coordinate systems do not address the above problem of dominance on tests being sparse (or becoming sparse in the course of search). As a coordinate system perfectly preserves dominance, whenever the dominance on tests is sparse, so it is in the dominance on the underlying objectives derived from them. Also, the number of underlying objectives can be still high, even for simple problems like the game of tic-tac-toe, and construction of a coordinate system is an NP-hard problem [64, 63].

derived objective

These observations call for alternative ways of efficiently translating an interaction matrix into a computationally tractable multi-aspect characterization of candidate solutions. In [112, 95] we came up with the idea of discovering *approximate* objectives by heuristic clustering of interaction outcomes. The proposed method DOC efficiently clusters an interaction matrix into a low number of performance measures, which we refer to as *derived objectives*, to clearly delineate them from the exact underlying objectives. By corresponding to a subset of tests, each derived objective captures a 'capability' that can be seen as a generalization of skills discussed earlier.

Technically, DOC replaces the conventional evaluation stage of the GP workflow (cf. Sect. 1.5.3) with the following steps:

1. *Calculation of interaction matrix.* We apply every program in the current population P, $|P| = m$, to every tests in T, $|T| = n$, and obtain so an $m \times n$ interaction matrix G (4.3).
2. *Clustering of tests.* We treat every column of G, i.e. the vector of interaction outcomes of all programs from P with a test t, as a point in an m-dimensional space. A clustering algorithm of choice is applied to the n points obtained in this way, and produces a partition $\{T_1, \dots, T_k\}$ of the original n tests in T into k subsets (clusters), where $1 \leq k \leq n$ and $T_j \neq \emptyset$.
3. *Calculation of derived objectives.* For each cluster T_j, we average row-wise the corresponding columns in G. The result is an $m \times k$ *derived interaction matrix* G', with the elements defined as follows:

$$g'_{i,j} = \frac{1}{|T_j|} \sum_{t \in T_j} g(p_i, t) \tag{4.10}$$

where p_i is the program corresponding to the ith row of G, and $j = 1, \dots, k$.

The columns of resulting G' matrix define the k derived objectives that characterize the programs in P in the context of the tests in T. The jth derived objective for a program p_i corresponding to the i row of the derived interaction matrix G' amounts to

$$f^j_{\textsc{doc}}(p_i) = g'_{i,j}. \tag{4.11}$$

Example 4.3. Figure 4.1 presents the example of DOC deriving objectives from a 4×5 interaction matrix G. The clustering algorithm partitions the tests into $k = 2$ clusters $\{t_1, t_2\}$ and $\{t_3, t_4, t_5\}$. Averaging the corresponding columns in G leads to the 4×2 derived interaction matrix G'. The graph plots the programs' positions in the resulting two-dimensional space of derived objectives. ■

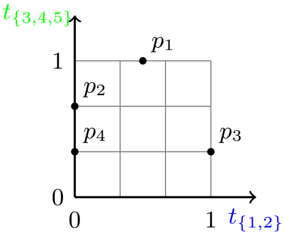
The derived objectives constructed by DOC form a compact, multi-aspect evaluation of the candidate solutions in P, and serve as a basis for selecting the most promising programs. Rather than devising an ad-hoc selection algorithm, it is natural to employ here multiobjective methods like NSGA-II [26]. Multiobjective selection allows programs that feature different behaviors (capabilities) coexist in a population, even if some of them are better than others on the conventional objective function f_o. In DOC, a capability can be identified with passing a specific group, or even *class* of tests. In case of the parity-3 problem illustrated in Sect. 2.3, a capability could be

G	t_1	t_2	t_3	t_4	t_5
p_1	1	0	1	1	1
p_2	0	0	1	1	0
p_3	1	1	0	1	0
p_4	0	0	1	0	0

(a)

G'	$t_{\{1,2\}}$	$t_{\{3,4,5\}}$
p_1	$1/2$	1
p_2	0	$2/3$
p_3	1	$1/3$
p_4	0	$1/3$

(b)



(c)

Fig. 4.1: An example of derivation of two search objectives from a matrix G of interactions between four programs $p_1, \ldots, p_4$ and five tests $t_1, \ldots, t_5$ (a). The tests (corresponding to the columns of G) are clustered into two clusters, marked in colors, according to a distance metric (here: Euclidean distance). The centroids of the clusters form the derived interaction matrix G' (b), in which each column defines a derived objective. The derived objectives form new objective space, with programs' locations shown in inset (c).

associated with passing all tests with the first input variable set to *true*. See Sect. 9.8 for a more detailed description of NSGA-II.

Derived objectives bear certain similarity to the underlying objectives discussed at the beginning of this section [16, 23, 64, 63]. However, as Example 4.3 and Fig. 4.1 show, they are not guaranteed to preserve dominance: new dominance relationships may emerge in the space of resulting derived objectives. For instance, given the interaction matrix as in Fig. 4.1a, program p_3 does not dominate p_4, however it does so in the space of derived objectives (Fig. 4.1c). As a result of clustering, some information about the dominance structure has been lost. This inconsistency buys us however a critical advantage: the resulting dominance relation is more dense and thus likely to impose a reasonably strong search gradient on an evolving population.

Although DOC may lead to dominance in the space of derived objectives where such relation was originally absent, in another work under review [113] we show formally that derivation of objectives will always preserve dominance if it already held for a pair of candidate solutions. Also, it cannot reverse the direction of dominance that already existed in the original space of outcome vectors.

Because clustering *partitions* the set of tests T (rather than only *selecting* some of them), none of the original tests is ignored in the evaluation process. In this sense, DOC tends to embrace the entirety of information available in an interaction matrix, which makes it different from and potentially more robust than methods that select tests, like [109] reviewed briefly at the end of Sect. 4.3. The more two tests are similar in terms of programs' performance on them, the more likely they are to end up in the same cluster and contribute to the same derived objective. In particular, tests characterized with identical outcome vectors are guaranteed to be included in the same derived objective.

The only parameter of the method is the number of derived objectives k. For $k = 1$, DOC degenerates to a single-objective approach: all tests form one cluster, and G' has a single column that contains solutions' evaluations as defined by (1.7), normalized by $|T_j|$ in (4.10). Setting $k = n$ implies $G' = G$, and every objective being derived from a single test. As we showed in [112], using k in the order of a few is most beneficial. Alternatively, the choice of k can be delegated to the clustering algorithm [95].

Similarly to IFS and CS, evaluation performed by DOC is contextual: all programs in P together determine the values of derived objectives. Objectives are derived independently in every generation of a GP run and are thus transient and incomparable across generations. This however does not prevent them from driving search more efficiently than conventional GP and IFS on most benchmarks, which we demonstrated in [95], and in coevolutionary settings, where T varies from generation to generation [112].

4.5 Summary

In the context of behavioral evaluation and execution record (Chap. 3), IFS, CS and DOC all rely on the same source of information for evaluation: an outcome vector resulting from the comparison of program output with the desired output (Fig. 3.2). In contrast to the conventional objective function f_o (1.7) that simply counts the zeroes (failed tests) in that vector for the program which is being evaluated, these methods require simultaneous access to outcome vectors of all programs in a population. Only then can they assess the subjective difficulty of tests (IFS), estimate the subjective odds for pairs of tests being simultaneously passed (CS), or group the tests into meaningful clusters to form derived objectives (DOC). In consequence, they will in general lead to different selection outcomes (see Examples 4.1, 4.2, and 4.3).

There is however more information available in an execution record and in the tests that define a program synthesis task. In particular, IFS, CS and DOC care only whether a test has been passed or not, and ignore *what* is the actual program output and the desired program output. These more detailed data open the door to more 'inquisitive' extensions of GP, with semantic GP presented in the next chapter being an important contemporary representative.