A Hierarchical Cooperative Evolutionary Algorithm

Shelly X. Wu
Computer Science Department
Memorial University of Newfoundland
St John's, Canada, A1B 3X5
xiaonan@mun.ca

Wolfgang Banzhaf
Computer Science Department
Memorial University of Newfoundland
St John's, Canada, A1B 3X5
banzhaf@mun.ca

ABSTRACT

To successfully search multiple coadaptive subcomponents in a solution, we developed a novel cooperative evolutionary algorithm based on a new computational multilevel selection framework. This algorithm constructs cooperative solutions hierarchically by implementing the idea of group selection. We show that this simple and straightforward algorithm is able to accelerate evolutionary speed and improve solution accuracy on string covering problems as compared to other EAs used in literature. In addition, the structure of the solution and the roles played by each subcomponent in the solution emerge as a result of evolution without human interference.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic methods

General Terms

Algorithms, Design, Experimentation

Keywords

Cooperative evolutionary algorithms, Multilevel selection, Hierarchical model, Emergent decomposition, Coevolution

1. INTRODUCTION

Evolution is based on fierce competition between individuals [6]. Evolutionary Computation (EC), which mimics natural evolution, also favors fitter individuals, and therefore is normally regarded as an optimization process on individuals. This implies that EC may fail to solve problems which require a set of cooperative individuals jointly to perform a computational task. Those individuals might be highly dependent on one another. From this perspective, EC should conduct not merely a multimodal search; the interactions between coadapted individuals need to be taken into account.

GECCO'10, July 7–11, 2010, Portland, Oregon, USA.

To complicate matters further, individuals may contribute differently in cooperation, and hence might carry unequal fitness. Weak individuals will be more likely to be eliminated from the population, despite their unique contributions in a collaboration. In order to provide reasonable opportunities for the emergence of cooperation through evolution, it is therefore necessary to consider extensions to the basic evolutionary computational models.

Cooperative CoEvolutionary Algorithms (CCEAs) [7] and Individual Evolutionary Algorithms (IEAs) [4] are two well-known extensions of classic EAs used for evolving cooperative solutions. Both algorithms address, in different ways, the issues of problem decomposition, interdependencies between subcomponents, credit assignment, and the maintenance of diversity, which according to Potter and de Jong [8] are essential in cooperative EAs. However, they lack flexibility in determining the structure of solutions. In both cases, optimization is defined on individuals but not on collaborations. In this paper, we propose a hierarchical cooperative EA based on a new computational multilevel selection framework. This novel algorithm constructs cooperative solutions hierarchically with the help of the group selection model proposed by Traulsen *et al.* [10] and the hierarchical model suggested by Banzhaf [1]. The former encourages cooperation by introducing competition between groups. The latter allows the algorithm to automatically control the number of levels in a hierarchical structure and the size of a collaboration. We investigated the performance of the algorithm on string covering problems, and compared it with a classic EA, a CCEA, and an IEA. Experimental results demonstrate that our algorithm not only improves accuracy but also accelerates evolution. Hierarchical structures emerge through the algorithm without human intervention.

The paper is organized as follows. Section 2 begins with the introduction of two cooperative EAs, CCEAs and IEAs, and multilevel selection (MLS) theories. Section 3 discusses string covering problems and data sets used in this contribution. Section 4 introduces a computational MLS framework, and explains the new algorithm in detail. Section 5 shows the experiments and obtained results. Section 6 concludes.

2. RELATED WORK

2.1 Cooperative Evolutionary Algorithms

Cooperative EAs assume a complex problem can be decomposed into simpler subproblems, and therefore solve the problem in a divide-and-conquer fashion. They extend the basic evolutionary computational model to support the evo-

lution of cooperation (see [8] for details). The discussion here will focus on two cooperative EAs: cooperative coevolution algorithms [7] and individual evolutionary algorithms [4], as they represent two principal ways to organize population.

CCEAs divide the population into multiple subpopulations, where each subpopulation is analogous to a species. Individuals in a species represent a subcomponent of a solution, and are evolved in their own subpopulation. To evaluate the fitness of an individual, a collaboration with representatives from each of the other species is formed. The fitness of the collaboration is evaluated, and is used as the fitness of the individual being evaluated. The collaborative fitness measures how good the collaboration is as a solution. Individuals who cooperate well with others will achieve high fitness.

IEAs are also known as Parisian approach. Their population is comprised of individuals which represent partial solutions to a target problem. A collaboration is formed by selecting n individuals from the population, where n is the number of subcomponents in a solution. In each generation, the performance of every single individual and the collaboration are evaluated by a local fitness function and a global fitness function, respectively. Periodically the local fitness values of individuals are adjusted based on the results of global fitness evaluations. In particular, the local fitness value of an individual is incremented or decremented after considering aspects such as: global fitness value of the collaboration, local fitness values of other individuals, as well as the individual's potential for improving the aggregate solution.

If we compare the two algorithms, on the one hand they both i) define individuals as partial solutions, and evolve them to locally built better subcomponents that jointly form better solutions; ii) form a collaboration as a complete solution and measure its performance by a fitness function. This sort of fitness function introduces evolutionary pressure for coadaption to occur; iii) consider credit assignments so that cooperative individuals with unique contributions to a solution are favored by evolution; iv) require preserving diversity in the population in order to maintain a set of complementary partial solutions. On the other hand, they lack flexibility in determining the structure of solutions. IEAs need to know *a priori* the number of subcomponents in a solution to determine the size of a collaboration. CCEAs can dynamically adjust the number of species (corresponding to sub-problems), but depend on an accurate definition of evolutionary stagnation. Normally evolution stagnates when the fitness of the best collaboration does not make a specified improvement over a certain number of generations. In addition, adding a new species will discard previous computational efforts, because evolution has to start over again. Furthermore, both algorithms neglect the optimization on collaborations. Since evolution is driven by individual fitness, in order to reflect individuals' contributions, CCEAs use collaborative fitness as individual fitness, which may cause the Red Queen effect [11], while IEAs have to introduce an extra step to redistribute the global fitness.

2.2 Group Selection and Multilevel Selection

In nature, the success of cooperation is witnessed at all levels of biological organization. A growing number of biologists have come to believe that the theory of group selection is the explanation, even though this theory has been unpopular for some decades [2]. Group selection models divide individuals into groups, where they only get to interact with members of the same group. Selection operates within groups and between groups. Within-group selection equals natural selection as understood commonly; it selects individuals in a group proportionally to fitness. Individuals, therefore, compete against each other in the pursuit of their own interests. Between-group selection, in contrast, examines the total productivity of groups, and prefers the group with the best performance or the group whose individuals cooperate best. It forces individuals to coadapt so that a cohesive group can be formed. It also resolves and reduces conflicts within groups, because conflicts would compromise group performance. In short, competition between groups encourages the emergence of cooperation within groups. In group selection models, individuals and groups are relative: groups can be regarded as individuals on a higher level, so that a new level of dynamics can act upon them. In this way, a hierarchical or nested structure can be constructed. This new perspective is now called multilevel selection (MLS) theory [10].

Competition between groups not only helps to construct hierarchies, but also accelerates evolution, as demonstrated by Banzhaf [1] through a series of experiments on a very simple artificial chemistry system. In that article, lower level entities are bonded together as a group by cooperative interactions. When such a group, which we can term a new entity, competes with less cooperative entities from lower-levels, it will take over the population at the end. The larger the difference among competing entities, the quicker the competition is settled; for example, the population with a group of 3 entities, and 4 autocatalysts converges faster than a population with 2 groups of 3 each, and 1 autocatalyst. The reason is that entities on different levels are allowed to compete against each other.

This design leads to a very interesting extension. Suppose the fitness of entities now no longer depends on their size, but rather on how they maximize a specific goal of a problem; therefore, a group on a higher level would not necessarily have a higher fitness. When groups on different levels compete with each other, groups with higher fitness regardless of their level will be favored by selection, hence are more frequently selected. In a sense, at which level the selection should act on is totally determined by the fitness of entities. We know evolution is parsimonious; higher level groups with lower fitness will be unstable, and will be eliminated from the population by competition; so the hierarchies will not grow exponentially, but stop at the most appropriate level required by the nature of the problem. Once the most stable level is decided, the best group structure will be found.

3. STRING COVERING PROBLEMS

String covering problems aim to discover a **set** of N binary strings that matches as strongly as possible another set of K binary strings, where K is typically much larger than N. The N and K binary strings are called match set (M) and target set (T), respectively. If every string in T has to be matched by at least one string in M, the strings in M must contain patterns shared by multiple target strings. Hence, the match set must have the capacity to generalize. However, on the other hand, we prefer the matches as strong as possible; so the match set also needs to be specific. In other

words, M must satisfy two contradictory criteria: generalization and specialization. The former means to minimize the size of M, and the latter means to maximize the match strength of M.

In this research, we generate four target sets by using the four schemata shown in Figure 1, respectively.

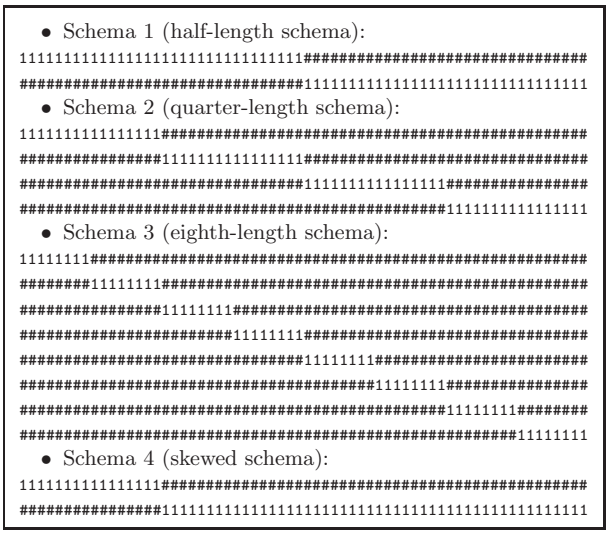


Figure 1: The four target sets used in this study are generated from above schemata.

Each schema contains at least two 64-bit string templates with a fixed region (marked by 1's) and a variable region (marked by #'s). Target strings created by a string template will share the same fixed region, but have randomly decided 0's and 1's in the variable region. For the first 3 schemata, 200 target strings are generated by each single string template; hence in total there are 400 target strings in target set 1, 800 in set 2 and 1600 in set 3. We create skewed data distribution for target set 4 by generating 200 strings from the first template and 20 from the second one.

Given any target set above, there is no overlap between patterns specified by fixed regions; so more than one match string needs to cooperate to fully cover a target set. Searching for several match strings simultaneously, or we say multiple coadaptive subcomponents in general, poses a challenge for classic EAs, as they always converge to a single optimum. Researchers, such as Forrest *et al.* [5, 9] and Potter *et al.* [8], investigated how to extend classic EAs to evolve cooperative solutions, and tested their algorithms on string covering problems to find all necessary match strings.

Our research also uses string covering problems as a test bed for the same purpose. Other reasons for selecting string covering are that it is simple (we are able to construct artificial string covering problems with known optima in different fitness landscapes) and practical (the experiment findings can be easily applied to similar application domains, such as other instances of set covering problems).

4. ALGORITHM DESIGN

The inspiration of this research comes from group selection theories and the hierarchical model in [1]. The former promotes the emergence of cooperation through evolution, and the latter is able to control the hierarchical structure in a solution. In this section, we will first show a computational multilevel selection framework that utilizes both ideas. Based on this framework, we will introduce a novel cooperative EA which automatically constructs cooperative solutions in a hierarchical way.

4.1 A computational MLS framework

Multilevel selection extends group selection from two levels to multiple levels. Adaptations now can potentially evolve at any level. When between-group selection dominates within-group selection, the groups resulting from one level may become the individuals for groups at the next level, as shown in Figure 2(a). Hence, MLS plays a role in explaining the transitions to new levels of hierarchical organization. This idea sheds some light on how cooperative solutions can be built incrementally in computational settings. However, when one applies this model computationally, immediate questions arise as to how to organize and define selection on groups, how many levels to use, which level to select, and how to bring those levels together.

Here we propose the computational multilevel selection framework of Figure 2(b) to address these questions. Our

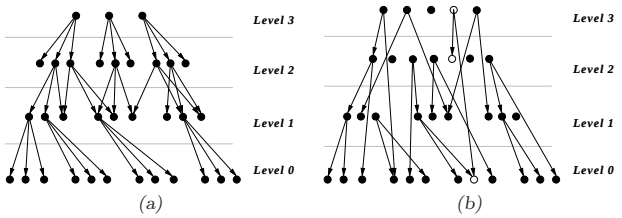


Figure 2: Comparison of two Multilevel Selection frameworks. (a) Framework by Chu et al. [3], in which competition takes place between groups on the same level; (b) Hierarchical framework, which considers selection not only between groups, but also between levels.

framework contains two types of entities. One is individuals, denoted by dots on level 0. Individuals are the most basic elements after problem decomposition. They are independent, without being aware of the collaborative goal. Apparently, there is no cooperation on this level. The other type of entities are groups, represented by dots on level 1 and higher. They are compositions of existing individuals or groups.

Initially, only individuals exist in the framework. Groups and new levels are created dynamically by a new operator called "**cooperation**". When cooperation applies, individuals and groups on all levels, if any, are mixed together, from which two entities, either groups or individuals, are selected proportional to fitness to form a new group. For example, as highlighted in Figure 2(b) by white circles, an individual on level 0 and a group on level 2 can cooperate to create a new group on level 3. This applies the idea from [1] but with an explicit fitness function added. Compared to the biological MLS framework, which considers competition between entities only on the same level, the new operator applies selection pressure not only between groups, but also between levels. One advantage of this scheme is to automatically decide the levels to select on; because selection is driven by the fitness of entities on different levels. A new level emerges only if groups on a particular level have an advantage in fitness; otherwise, the new level will be removed by competition between levels. As a result, the structure of the hierarchy develops to the most appropriate level. We keep crossover and mutation as two further operators, which

are able to introduce new groups at all levels. These three operators push evolution forward on group levels.

Individuals also need to be optimized locally to build better groups. To this end, a group is first selected proportional to fitness from groups on all levels; an individual is then selected from this group as a parent. Parents cross over or mutate to produce new individuals. Obviously, the survival of an individual is now associated with the performance of its group. Individuals have to specialize on different roles in a group to strive for a high group fitness, even though such kind of roles are not assigned or unknown prior to runs. We use Traulsen's group selection model [10] instead of Wilson's [12], as it is more robust against parameter changes [13]. As required by this model, groups will not mix and reform during the evolution.

4.2 The algorithm

The algorithm that implements the above framework is shown in Figure 3. In a first step (initialization) N_1 individuals are randomly generated and have their fitness evaluated.

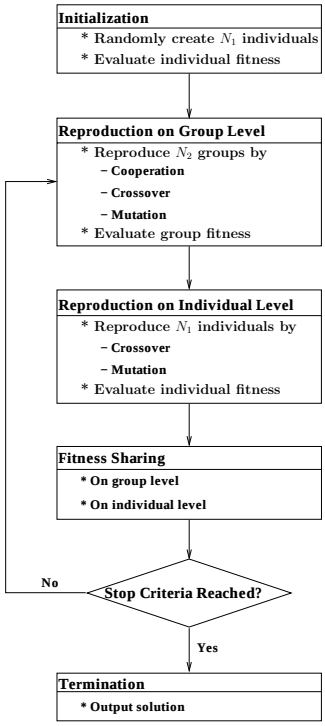


Figure 3: Outline of the hierarchical evolutionary algorithm with N_1 individuals and N_2 groups.

Reproduction on group levels produces N_2 new groups every generation by applying three evolutionary operators, i.e. cooperation, crossover and mutation, with a user-defined probability. Entities involved in these operators are selected by the roulette-wheel selection, but from different sources. Cooperation selects participants from both individuals and groups (i.e. from level 0 and above), while the other two operators select only from groups (i.e. from level 1 and above). Cooperation composes a new group by adding individuals from the two selected entities; crossover exchanges individuals in two groups starting from a randomly selected position in each group; and mutation randomly adds, removes, or replaces an individual in a group (for simplicity, we only consider removing individuals in the following experiments). Once a new group is created, its group fitness and its validation are evaluated. A group is valid only when every member has a unique contribution towards the cooperative goal. This is to guarantee there are no free riders in a group, so group size will not increase unnecessarily. New groups and groups in the current generation compete for N_2 positions in the next generation.

To produce offspring on the individual level, we first apply group selection to select parent individuals for crossover and mutation. If no groups are currently available in the population, parents are selected directly from the individual level based on fitness. Once parents are selected, two-point crossover and bit-flip mutation are conducted. The fitness of new individuals is evaluated. Only the best N_2 individuals among the new offspring and individuals in the current generation are kept for the next generation.

Please note that this algorithm evolves N_1 individuals and N_2 groups separately. We keep a constant number of individuals in the population, simply because individuals are the most basic building blocks. Only when individuals have fully exploited their local environment, or have maximized their fitness, will it be possible to find optimal groups.

The probability of conducting cooperation and crossover is controlled by a user-defined parameter, but the probability for mutation is decided by an entity's fitness: it is inversely proportional to the fitness, such that entities with lower fitness experience larger changes to their genotypes.

Preserving diversity is mandatory on the individual and group levels, because the algorithm needs to maintain a set of different partial solutions so that all required subcomponents can present in the final solution. Various niching mechanisms can be used, such as crowding, fitness sharing, implicit sharing, or even user-defined niching schemes.

The above steps will be repeated until a predefined termination criterion is reached, e.g. the maximum generation, a desired fitness or accuracy.

In summary, we proposed a novel hierarchical EA to search multiple coadaptive subcomponents in a solution; this algorithm extends classic EAs by introducing group selection and the evolution on group levels. Group selection favors individuals who cooperate and contribute in a group. It avoids manually linking roles of individuals to their fitness, which unfortunately is a mandatory step in IEAs. It also overcomes the Red Queen Effect in CCEAs, because the performance of both individuals and groups are measured. The evolution on group levels optimizes groups, which in turn should accelerate evolution on the individual level. We expect the algorithm will evolve faster and find better solutions. In addition, because of the selection between levels, this algorithm is able to build solutions hierarchically, and decide the most appropriate level of hierarchies and the size of a collaboration without human interference.

5. EXPERIMENTS

In order to test whether the Hierarchical EA (HEA) can indeed build solutions hierarchically from smaller subcomponents, we ran experiments on the four target sets for the string covering problems discussed in Section 3. For each set, we expect the algorithm to return a match set whose match strings have the exact patterns as the string templates in the corresponding schemata. We compare the results with

the ones produced by the three control algorithms: a classic EA, a CCEA, and an IEA.

5.1 Representation, Fitness Function and Fitness Sharing

Individuals are represented as 64-bit strings on the alphabet {1, *}, not {1, 0} used in previous work [5, 8, 9]. "*" is a "don't care" symbol representing either "1" or "0" on a position, whose value is not shared by most strings in a target set. This change allows us to easily tell the accuracy of solutions and the level of generality.

We define the individual fitness function as the following:

$$f(x) = \alpha \times Ratio_1(x) \times Coverage_{idv}(x), \quad (1)$$

where $Ratio_1(x)$ shows the percentage of the number of 1's in the representation of individual x, $Coverage_{idv}(x)$ the string coverage of individual x, calculated by the number of target strings covered by individual x over the cardinality of the target set, and α is a weighting coefficient. Basically, the individual fitness function is looking for a specific individual (individuals with more 1's), but at the same time with high string coverage.

Once we know how the performance of individuals is measured, our intention of conceiving the four different target sets is obvious. For the first three target sets, solutions are becoming harder to find as the number of subcomponents increases and fixed regions become progressively shorter with respect to the variable regions of the string templates. Target set 4 is the only set whose solution contains subcomponents with **unequal** fitness, because of the unequal length in the fixed regions and its skewed data distribution. Therefore, target set 4 presents a more difficult problem than the others.

The group fitness is defined as

$$g(y) = \frac{\sum_{i=0}^{n} f(x_i)}{n} \times Coverage_{grp}(y), \quad (2)$$

which considers average individual fitness in group y and string coverage, $Coverage_{grp}$, of group y, where n is the group size. Group fitness favors groups whose individuals cooperate to provide maximum coverage, while each individual is optimized to specialize its role in the cooperation.

We apply fitness sharing on the individual level to explicitly preserve diversity. Two individuals have to share their fitness if their genotypic distance is within the range of a sharing radius. On the group level, we reward groups who provide new string coverage that does not appear in the coverage of the best group from the previous generation; we also penalize groups who share coverage with others.

5.2 Experimental setup

We ran HEA and the three control algorithms on a PC with an AMD $Turion^{TM}$ 64×2 CPU at 1.6GHz and with 2 GB of RAM. The parameter settings are shown in Table 1. The four target sets are denoted as ts1, ts2, ts3, and ts4, respectively.

We measure the performance of all algorithms by convergence time and average number of mismatched bits. The convergence time is the number of seconds an algorithm needs to find the best solution. In our evaluations it is a better indicator than the number of fitness evaluation in reflecting evolutionary speed, given the fact that the algorithms conduct fitness evaluation differently (e.g. CCEAs only consider fitness on the collaboration level), which cause various amounts of time to complete. To calculate the average number of mismatched bits, we first count the number of different bits between each string template used by a target set and the closest match string returned by an algorithm, and then average over all string templates.

Table 1: Parameter settings

Parameter	Classic EA	CCEA	IEA	HEA
Run	50	50	50	50
Generation	1000	1000	2000	2000
Number of groups	N/A	N/A	1	10
Cooperation rate	N/A	N/A	N/A	0.5
Crossover rate	0.95	0.95	0.95	0.95
Mutation rate	0.05	0.05	0.05	N/A
Group size	N/A	ts1/4:2	ts1/4:2	N/A
		ts2:4	ts2:4	
		ts3:8	ts3:8	
Fitness coefficient	ts1/4:4	N/A	ts1/4:4	ts1/4:4
	ts2:16		ts2:16	ts2:16
	ts3:64		ts3:64	ts3:64
Niching radius	N/A	N/A	ts1/2:0.7	ts1/2:0.7
			ts3:0.5	ts3:0.5
			ts4:0.9	ts4:0.9

5.3 Evaluating HEA and Control Algorithms

We ran HEA and the three control algorithms on all four target sets. Table 2 shows average performance of the algorithms over 50 runs. Standard deviation of the convergence time and of average number of mismatched bits are enclosed in brackets. In order to allow a fair comparison, given the same target sets, all algorithms have the same amount of individuals in their population, as this number has direct implications for the evolutionary speed and solution quality. Species size in CCEA is calculated by dividing the total population size by the number of species. We also ran another set of experiments on CCEA using this number as the size of species, and kept the results for reference.

For target set 1 and 4, HEA always found a match set with **2** match strings, which perfectly matched all target strings; in other words, there was no mismatch in either case. However, it took a longer time for runs to converge on target set 4 than for runs on target set 1, because exploring and maintaining multiple match strings with unequal fitness is more difficult. For target set 2 and 3, all match sets returned by HEA contained **4** and **8** match strings, respectively. Because both sets obtained low average number of mismatched bits and relatively high standard deviation, we further collected the median as an extra evidence, which was 0 on both sets. The three numbers together indicate that most runs returned correct match sets. Convergence time on the two target sets was longer because the difficulty of the problem increased.

We now compare the four algorithms on the first three target sets. The classic EA can only find one match string out of many, as all match strings for a target set are equally good in terms of specialization and coverage. That algorithm fails the task with no surprise, because we know that

Table 2: Performance of four algorithms on target set 1, 2, 3, and 4.

Algorithms	*Target set 1*			*Target set 2*		
	# of Idv.	Convergence Time	Mismatch Bits	# of Idv.	Convergence Time	Mismatch Bits
HEA	**20**	**1.539(0.509)**	**0.000(0.000)**	**20**	**10.828(3.239)**	**0.095(0.256)**
Classic EA	20	0.595(0.169)	32.00(0.000)	20	2.140(1.113)	24.017(0.207)
CCEA[1]	20(10×2)	1.658(0.289)	0.617(0.462)	20(5×4)	19.664(9.246)	0.758(0.350)
CCEA[2]	40(20×2)	2.236(0.397)	0.600(0.536)	80(20×4)	23.132(7.429)	0.592(0.282)
IEA	20	3.171(1.091)	0.717(0.429)	20	7.560(1.529)	0.708(0.378)
Algorithms	*Target set 3*			*Target set 4*		
	# of Idv.	Convergence Time	Mismatch Bits	# of Idv.	Convergence Time	Mismatch Bits
HEA	**40**	**20.665(6.801)**	**0.088(0.145)**	**20**	**3.020(1.664)**	**0.000(0.000)**
Classic EA	40	12.743(3.717)	12.225(5.298)	20	0.350(0.372)	32.000(0.000)
CCEA[1]	40(5×8)	289.060(57.951)	0.950(0.270)	20(10×2)	5.258(2.170)	1.983(0.517)
CCEA[2]	320(40×8)	367.540(144.319)	0.467(0.183)	40(20×2)	5.970(2.839)	1.867(0.472)
IEA	40	59.323(18.991)	0.504(0.268)	20	1.981(0.987)	0.983(0.160)

it has a strong tendency to converge. Given same population size, CCEA outperforms IEA only on simple target sets, but not on hard ones. It cannot beat HEA on any of the three sets because of two limitations. First, CCEA does not maintain diversity within species; the way fitness is defined only helps to preserve diversity between species. In our experiments, the algorithm converged at generation average 88.8 for target set 1 given 40 individuals, at generation average 125.467 for set 2 on 80 individuals, and at generation average 263.226 for set 3 on 320 individuals. Once species have lost their diversity, the algorithm stops exploring the search space. Second, individual fitness depends on exactly who is in a collaboration, so it does not accurately measure the performance of individuals. As a result, the search will drift to suboptimal solutions. Increasing population size, though improving accuracy somewhat, provides little help to overcome these limitations. HEA also performs better than IEA on all test runs. The difference between the two algorithms is the choice of group selection. IEA only composes a single group by selecting the best n individuals from the population (where n is the group size), while HEA forms more than one group with various sizes and compositions, and considers the evolution on group levels. From this perspective, it increases the possibility of finding a solution faster.

Target set 4 is a new set we introduced in this study, which requires algorithms to optimize and maintain multiple subsolutions with unequal fitness simultaneously. It has proven to be the hardest one among the four sets; the classic EA converged to a string with 48 1's despite the low fitness, as searching for such a string is much easier than searching for a string with 16 1's; CCEA and IEA both obtained the highest average number of mismatched bits on this set. In contrast, HEA found a perfect match set very quickly.

Table 3 shows the statistical comparison of HEA over the two control algorithms on convergence time and average mismatch bits, using the two-tailed t-test with 98 degrees of freedom at a 0.05 level of significance. Since the p-value is less than 0.05 (except the convergence time of CCEA[1] on ts1), we can conclude that HEA achieved a significant improvement on accuracy and evolutionary speed when compared to the control algorithms on string covering problems.

5.4 Looking inside of HEA

In order to get a better idea why HEA evolves faster and finds more accurate solutions than the other cooperative

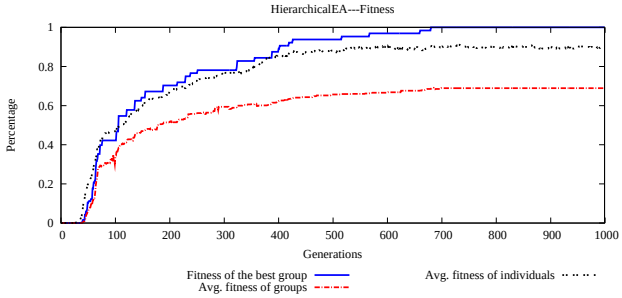
Table 3: The T-test results between HEA and the two control algorithms.

Target set		CCEA[1]	CCEA[2]	IEA
ts1	Time	0.227	4.807E-07	6.287E-08
	Mismatch	8.053E-08	7.061E-07	4.802E-10
ts2	Time	2.346E-05	2.499E-09	1.787E-05
	Mismatch	2.758E-10	2.155E-07	3.568E-08
ts3	Time	1.00E-21	8.007E-14	3.286E-12
	Mismatch	3.156E-16	1.815E-10	2.741E-08
ts4	Time	5.108E-05	6.856E-05	0.015
	Mismatch	4.218E-19	1.891E-19	8.405E-25

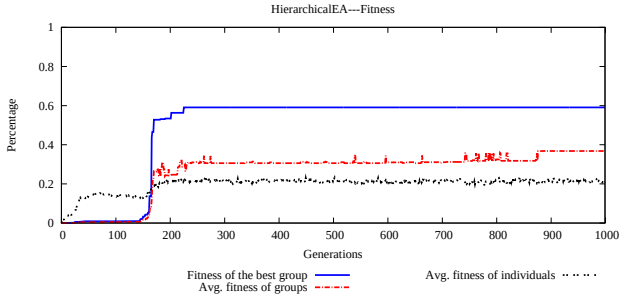
EAs, we investigate the algorithm by examining its performance in a typical run on target set 2 and 4. We choose these two sets because they represent two different situations, namely equal and unequal fitness of subcomponents in a solution. Target set 1 and 3 are not discussed here because they share the same features with target set 2.

Fitness is always a good place to start investigations as it reflects how evolution proceeds. Figure 4(a) and (b) depict fitness related information in a typical run on target set 2 and 4, respectively. We show the fitness of the best group, the average fitness of individuals and the average fitness of groups. Individual fitness and group fitness by definition are affected by coverage and specialization (the number of 1's in the representation). Therefore, we also show the average specialization of individuals and of the best group in Figure 4(c) and the coverage of the best group in Figure 4(d).

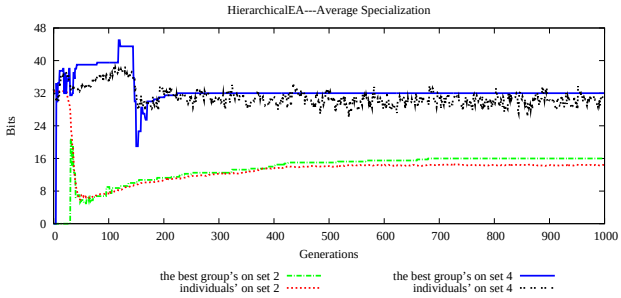
As we can see clearly in Figure 4(a) and (b), average individual fitness and group fitness improve steadily due to the evolution happening on individual and group levels. As a result, the fitness of the best group increases constantly on both sets. To be more specific, HEA optimizes the coverage first (see Figure 4(d)), because increasing the coverage will improve both individual and group fitness. However, the different properties of the two sets cause HEA behaving differently at this stage. Individuals are randomly generated at the initialization, so their average specialization on both sets is around 32 at the outset (see Figure 4(c)). For target set 2 which requires 16 1's in all match strings, the specialization has to drop in order to maximize the coverage. For target set 4, the specialization is increased first to optimize the coverage of the match string with 48 1's, and then decreased to optimize the coverage of the one with 16 1's. After coverage



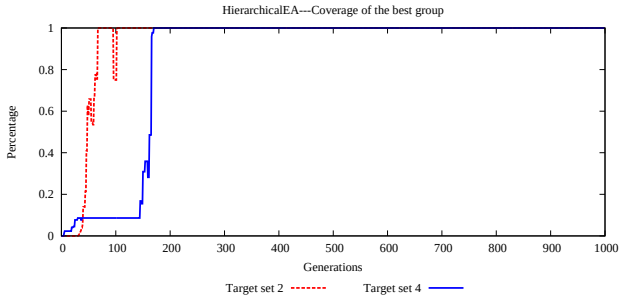
(a) Fitness changes on target set 2.



(b) Fitness changes on target set 4.



(c) Average specialization on target set 2 and 4.



(d) Coverage of the best group on target set 2 and 4.

Figure 4: A typical run on target set 2 and 4.

hits 1 (i.e. coverage has been optimized), HEA then turns to optimize the second part of the individual fitness, so we see that the specialization on both sets increases.

The run on target set 4 demonstrates very well the contributions of group selection to encourage cooperation, regardless of individuals fitness. As shown in Figure 4(c), during the first 200 generations, no matter whether the average specialization of the best group is moving towards 48 or 16, the average specialization of individuals always keeps a distance. This implies that even though HEA optimizes the match string with 48 1's in the first place, a few individuals covering target strings with 16 1's have managed to stay in the population. Such individuals provide new coverage to their group (i.e. they increase group fitness), hence the group and the individuals inside are more frequently selected and optimized. Therefore, they gradually dominate the population (the average specialization of the best groups and individuals begin to drop). Similarly, after the coverage hits 1, the specialization of individuals with 48 1's continues to increase, despite their low fitness and the domination of high fitness individuals with 16 1's.

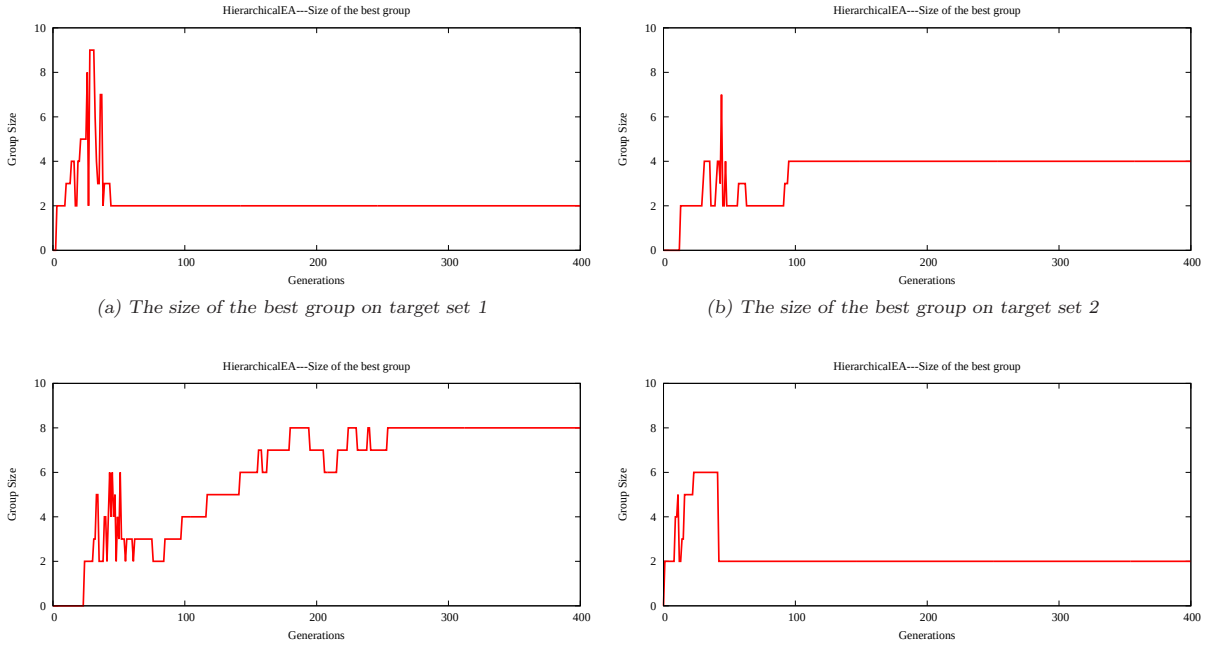
The process of searching for the structure of a solution on the four target sets is shown in Figure 5. We can easily see that the HEA is able to return a solution with the correct number of subcomponents, even though that number was not known *a priori*. Driven by the between-level selection introduced in [1], groups are maintained in the population if they show advantages in fitness; otherwise, they are eliminated. Therefore, we observe the size of the best group changing till the best size is found. We also notice that the group size fluctuates at the beginning of the evolution. This is because individuals during that time have similar coverage and fitness; small changes on group composition and size easily affect the group fitness.

6. CONCLUSIONS

In this paper, we discussed a hierarchical evolutionary algorithm to encourage cooperation through evolution. We claim the following contributions. First, the algorithm is based on a new multilevel selection framework that suits computational needs. Second, it applies Traulsen's group selection model in biology to promote cooperation. Third, a new operator called "cooperation" is introduced to construct the hierarchical structure in solutions. Finally, it uses the hierarchical model in [1] to control the structure. The new algorithm is investigated and compared with three control algorithms on string covering problems whose fitness landscapes have multiple equal or unequal fitness peaks. Based on our experiments, we conclude that the new algorithm improves both solution accuracy and evolutionary speed. In addition, the structure of a solution and the roles played by their subcomponents emerge as a result of evolution, rather than being designed by hand. In the future, we plan to study the evolutionary dynamics of this algorithm further and to tackle real-world problems that require a substantial degree of cooperation.

7. ACKNOWLEDGMENTS

W.B. would like to acknowledge support from NSERC Discovery Grants, under RGPIN 283304-07. The authors would also like to thank anonymous reviewers for their helpful comments and suggestions.



(a) The size of the best group on target set 1

(b) The size of the best group on target set 2

(c) The size of the best group on target set 3

(d) The size of the best group on target set 4

Figure 5: Hierarchically finding subcomponents in the solution for all target sets.

8. REFERENCES

[1] W. Banzhaf. On the dynamical of competition in a simple artificial chemistry. *Nonlinear Phenomena in Complex Systems*, 5(4):318–324, 2002.

[2] M. E. Borrello. The rise, fall and resurrection of group selection. *Endeavour*, 29(1):43–47, March 2005.

[3] D. Chu and D. J. Barnes. Group selection vs multi-level selection: Some example models using evolutionary games. In *IEEE Congress on Evolutionary Computation (CEC '09)*, pages 808–814, 2009.

[4] P. Collet, E. Lutton, F. Raynal, and M. Schoenauer. Individual GP: an alternative viewpoint for the resolution of complex problems. In W. Banzhaf, J. Daida, and et al., editors, *Proceedings of the 1st Genetic and Evolutionary Computation Conference (GECCO '99)*, volume 2, pages 974–981, Orlando, Florida, USA, 1999. Morgan Kaufmann.

[5] S. Forrest, R. E. Smith, B. Javornik, and A. S. Perelson. Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, 1(3):191–211, 1993.

[6] M. A. Nowak. Evolutionary dynamics of cooperation. In *Proceedings of the International Congress of Mathematicians*, pages 1523–1540, Madrid, Spain, 2006. European Mathematical Society.

[7] M. A. Potter and K. A. de Jong. A cooperative coevolutionary approach to function optimization. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature (PPSN III), Jerusalem, Israel*, volume 866/1994 of *LNCS*, pages 249–257. Springer Berlin/Heidelberg, 1994.

[8] M. A. Potter and K. A. de Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[9] R. E. Smith, S. Forrest, and A. S. Perelson. Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation*, 1(2):127–149, 1993.

[10] A. Traulsen and M. A. Nowak. Evolution of cooperation by multilevel selection. In *Proceedings of the National Academy of Sciences of the USA (PNAS)*, volume 103, pages 10952–10955. National Academy of Sciences, 2006.

[11] R. P. Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, George Mason University, Fairfax, Virginia, 2003.

[12] D. S. Wilson. A theory of group selection. In *Proceedings of the National Academy of Sciences of the USA (PNAS)*, volume 72, pages 143–146. National Academy of Sciences, 1975.

[13] S. X. Wu and W. Banzhaf. Investigations of Wilson's and Traulsen's group selection models in evolutionary computation. In *Proceedings of the 10th European Conference on Artificial Life, Budapest, Hungary, 2009*. LNCS, Springer, in press, 2010.