Cooperative Co-evolutionary Optimization of Software Project Staff Assignments and Job Scheduling

Jian Ren[1], Mark Harman[1], and Massimiliano Di Penta[2]

[1] Department of Computer Science, University College London, UK
[2] Department of Engineering, University of Sannio, Italy

Abstract. This paper presents an approach to Search Based Software Project Management based on Cooperative Co-evolution. Our approach aims to optimize both developers' team staffing and work package scheduling through cooperative co-evolution to achieve early overall completion time. To evaluate our approach, we conducted an empirical study, using data from four real-world software projects. Results indicate that the Co-evolutionary approach significantly outperforms a single population evolutionary algorithm. Cooperative co-evolution has not previously been applied to any problem in Search Based Software Engineering (SBSE), so this paper reports the first application of cooperative co-evolution in the SBSE literature. We believe that co-evolutionary optimization may fit many applications in other SBSE problem domains, since software systems often have complex inter-related subsystems and are typically characterized by problems that need to be co-evolved to improve results.

Keywords: Cooperative Co-Evolutionary Algorithm, Staff Assignments, Work Package Scheduling, Software Project Planning, Search Based Software Engineering.

1 Introduction

Software project management has been the subject of much recent work in the SBSE literature. Previous work has investigated the project staffing and planning problem either as a single-objective problem, or as a multi-objective problem in which the multiple objectives are, to some degree, conflicting objectives [3,4,13]. In this paper we introduce an alternative approach based on the use of a Cooperative Co-Evolutionary Algorithm (CCEA). We believe that a Cooperative Co-Evolutionary approach to project management is attractive because it allows us to model a problem in terms of sub problems (e.g., in project scheduling and staffing, the allocation of work packages to teams and allocation of staff to teams). These sub problems can be inter-related, but separate problems, for which the overall solution depends on the identification of suitable sympathetic sub-solutions to each of the subproblems.

We show how the two primary features of a project plan—i.e., the allocation of staff to teams and the allocation of teams to work packages—can be formulated as two populations in a Cooperative Co-evolutionary search. Co-evolution has been previously used in SBSE work [1,6,7], but all previous approaches have used *competing* subpopulations; the so-called predator–prey model of Co-evolution. In this paper, we adopt the alternative approach to co-evolution; Cooperative Co-evolution, in which the subpopulations

M.B. Cohen and M. Ó Cinnéide (Eds.): SSBSE 2011, LNCS 6956, pp. 127–141, 2011.

work symbiotically rather than in conflict with one another. We believe that this form of co-evolution may also find many other applications in SBSE work, since many Software Engineering problems are characterized by a need to find cooperating subsystems that are evolved specifically to work together symbiotically.

We implemented our approach and evaluated it on data from four real world software projects from four different companies, ranging in size form 60 to 253 individual work packages. We report the results on the efficiency and effectiveness of our approach, compared to a random search and to a single population approach. Our results indicate that the co-evolutionary approach has great promise; over 30 runs for each approach, co-evolution significantly outperforms both random and single population approaches for the effectiveness of the project plans found, while it also appears to be at least as efficient as a single population approach.

The paper makes two primary contributions: (1) The paper introduces a novel formulation of the Software Project Planning Problem using Cooperative Co-evolution and, to the best of our knowledge, this is the first paper in the SBSE literature to use cooperative co-evolution. (2) The paper reports the results of an empirical study with an implementation of our co-evolutionary approach, compared to random and single population evolution. The obtained results provide evince to support the claim that cooperative co-evolution is more efficient and effective than single population evolution and random search.

2 Problem Statement and Definitions

This section describes the problem model for the work packages scheduling and staff assignment problem in detail and addresses the use of the CCEA.

Finding an optimal work package scheduling for a large project is difficult due to the large search space and many different considerations that need to be balanced. Also, finding an optimal way to construct its project teams is crucial as well. In this paper, we focus on team construction with regard to team size.

In order to formulate this problem into a model, we make the following assumptions to simplify the problem: (1) staff members are identical in terms of skills and expertise, and staff only work on one team during the whole project, (2) WPs are sequentially distributed to teams, but they may still be processed at the same time, and (3) only one kind of dependency is considered: Finish-to-Start (FS). All three assumptions were found to be applicable to the four projects studied in this paper, all of which are real world software projects and therefore, though limiting, our assumptions do not preclude real world application.

2.1 Ordering/Sequence of Work Packages

To model the work needed to complete a project, we decompose the project according to its Work Breakdown Structure (WBS). WBS is widely used as a method of project decomposition. In a given WBS, the whole project is divided into a number of l small Work Packages (WPs): $\mathbf{WP} = \{wp_1, wp_2, \cdots, wp_l\}$. Two attributes of a WP, wp_i, are considered in this paper: (1) the estimated effort, e_i, required to complete wp_i,

and (2) the WP predecessor(s), dep_i, which need to be completed before wp_i can start to be processed. The estimated efforts for all WPs are represented as a vector: $\mathbf{E} = \{e_1, e_2, \cdots, e_l\}$, e.g.: wp_i requires e_i person-days to complete; and dependence information is represented as a two-dimensional vector as: $\mathbf{Dep} = \{dep_1, dep_2, \cdots, dep_l\}$ where $dep_i = \{wp_j, \cdots, wp_k\}$ if the predecessors of wp_i are wp_j, $\cdots$, and wp_k.

The order in which the WPs are considered is represented as a string, shown in Figure 1, where the WP ordering in the string indicates a specific sequence for distributing the WPs to project teams. Constraints of precedence relationships are satisfied as each is processed, with the effect that a project cannot start until its dependent WPs have been completed.

Distributing Order:	$1st$	$2nd$	$3rd$	$\cdots$	$(l-1)th$	l-th
Work Package ID:	3	2	6	$\cdots$	l	$l-4$

Fig. 1. WPO Chromosome: The gray area is the representation of the solutions for the ordering for distributing a set of l work packages. A solution is represented by a string of length l, each gene corresponding to the distributing order of the WPs and the alleles, drawn from $\{1, ..., l\}$, representing an individual WP.

2.2 Staff Assignments to Teams

A total of n staff are assigned to m teams to execute the WPs. The size of each team (their 'capacity') is denoted by a sequence $\mathbf{C} = \{c_1, c_2, \cdots, c_m\}$.

Staff:	S_1	S_2	S_3	$\cdots$	S_{n-1}	S_n
Assigned To Team No.:	2	4	3	$\cdots$	m	3

Fig. 2. TC Chromosome: The gray area is the representation of the solutions for Team Construction or the assignments of a set of n staff to a set of m teams. A solution is represented by a string of length n, with each gene corresponding to a staff and the alleles, drawn from $\{1, ..., m\}$, representing assignment of the staff.

2.3 Scheduling Simulation

We use a single objective fitness evaluation for both populations in our co-evolutionary approach, i.e., the project completion time. The processing of the WPs by the teams is simulated by a simple queuing simulation as described in previous work [13,20]. In this approach, the WP dependence constraints are satisfied by arranging the order in which WPs are assigned to teams. However, we want to avoid the order in which the successor wp_i is right after its predecessor wp_j. In such a case, wp_i has to wait until wp_j is finished before it can be distributed to an available team. There are two ways for the managers to minimize the team's unused available time: 1) interlacing: managers can choose to insert one or more WPs between wp_i and wp_j so when wp_i is waiting for wp_j those inserted WPs can keep all the teams functioning, or 2) using mitigation: distribute the predecessor wp_j to a team with the highest possible capacity, so that the completion time of the predecessor is the shortest, and therefore, the wait time of wp_i is mitigated

to be the shortest one. In our case, we simply rely on the search based algorithm that, by producing different WP orderings, can enact both interlacing or mitigation. Further details about the simulation of WP scheduling can be found in a previous work [13].

3 Optimization Method: Cooperative Co-evolutionary Algorithm

The Cooperative Co-Evolution Algorithm (CCEA) [21] was proposed to solve large and complex problems by implementing a divide-and-conquer strategy. CCEA was originally designed to decompose a high-dimensional problem into smaller sub-problems that could be handled by conventional evolutionary algorithms [24]. Using CCEA, the individuals from each population represent a sub-solution to the given problem. To search for a solution, the members of each population are evolved independently, and interactions between populations only occur to obtain fitness.

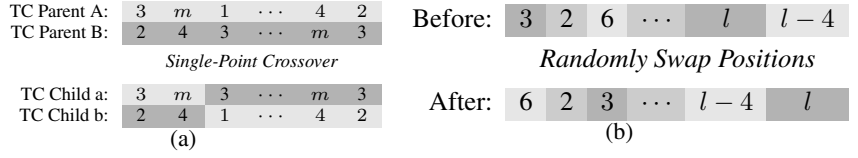


Fig. 3. (a) TC Single Point Crossover and (b) WPO Mutation

3.1 Solution Representations and Genetic Operators

There are two species of solutions in this evolutionary process: One (WPO) contains solutions representing the ordering in which WPs are distributed to teams, and the other (TC) represents the Team Constructions, i.e., the number of staffing persons for each team.

For solutions representing staff assignments or TC, as shown in Figure 2, we encoded the solutions in the following format. The assignment of a set of n staff to a set of m teams is represented as a string of length n. Each gene of the chromosome corresponds to a staff member. The alleles ranging from 1 to m represent the team that the staff is assigned to. A single-point crossover is performed for the TC specie. Basically, the child takes half of chromosome from each of both parents. The mutation operator is thus set to assign each staff randomly to another team.

To achieve a fair comparison between projects, we chose the number of staff $n = 50$ and the number of working teams $m = 5$. Also, we verify every new generated solution of TCs before evaluating it with the fitness function, to ensure in each solutions every team has at least one staff member. This "no empty team" check is required because a team can be empty during the evolutionary process if all staff members are assigned to other teams. The representation of WP ordering is shown in Figure 1. The crossover operator for such a representation is explained in [13], while the mutation operator randomly swaps a WP to another position in the queue, with a mutation rate of 0.2 per gene, calibrated after trying other (higher and lower) rates.

To satisfy the dependency constraints among WPs, a dependency check is required. Solutions that violate the dependency constraints are "repaired" as explained in Section 2.3.

3.2 Initial Populations

The initial populations are randomly generated and subject to satisfy the of "no empty team" rule and dependency constraints among WPs. The population size is set to 50 for both species.

3.3 Termination Condition

We experimented with 3 sets of configurations formed of the Internal (I) and External (E) number of generations for CCEA. The number of internal generations relates to the evolution of each sub-population, while the number of external generations represents how many times each population provides an updated reference to help the other population co-evolve. As shown in Table 1, these 3 configurations are all allowed the same budget of fitness evaluations. Although all these three configurations evolve solutions in both TC and WPO populations for the same number of generations, the level of communication between the two populations varies.

Table 1. Three sets of configurations for CCEA each of which requires the same total number of evaluations before it is terminated

Config.	Int_Gen	Ext_Gen
I	1	100
II	10	10
III	100	1

For instance, with Config. I, CCEA evolves solutions in both populations for a single generation only (internal generation) and then provides the updated individuals for fitness evaluations of the other population. Finally, before the evolution process terminates, the TC population provides an updated reference for the WPO population for a total of 100 iterations (external generation), and vice versa. Config. III is not a CCEA by definition because during the whole period of the cooperative co-evolutionary process, the communication only happens only once. Therefore Config. III is a 'non-co-evolutionary' approach, against which we compare the other (co-evolutionary) approaches. By implementing the non co-evolutionary approach as a 'special case' (by suitable choice of parameters) we remove one source of possible bias that would otherwise result from experimenting with two totally different implementations: one co-evolutionary and the other not.

4 Empirical Study

The goal of this empirical study is to compare our new CCEA approach with a non-co-evolutionary genetic algorithm and a random search. We study the effectiveness and efficiency of our approach, alternatives and (for purely 'sanity check' purposes) random search on four industrial projects, named *Projects A, B, C and D*, described below and for which quantitative data are summarized in Table 2.

Table 2. Characteristics of the four industrial projects

Projects	#WPs	#Dependencies	Total Effort
A	84	0	4287 (person-hours[1])
B	120	102	594 (person-days)
C	253	226	833 (person-days)
D	60	57	68 (person-days)

Project A is a massive maintenance project for fixing the Y2K problem in a large financial computing system from a European organization. According to its WBS, the application was decomposed into WPs, i.e., loosely coupled, elementary units subject to maintenance activities. Each WP was managed by a WP leader and assigned to a maintenance team. No WP dependency was documented, and thus, no constraint had to be satisfied in *Project A* scheduling. *Project A* can be considered as representative of massive maintenance projects related to highly-standardized tasks such as currency conversion, change of social security numbering, etc. *Project B* aimed at delivering the next release of a large data-intensive, multi-platform software system, written in several languages, including DB II, SQL, and .NET. *Project C* aimed at delivering an online selling system to provide North American businesses of all sizes with a fast and flexible way to select, acquire and manage all their hardware and software needs. The system includes the development and testing of website, search engine, and order management, etc. *Project D* is a medium-sized project implemented in a large Canadian sales company. This project aims to add new enhancement to the supply chain of the company by allowing instant and on-demand conversion of quotes to orders. This change is both internal and customer facing and ultimately affects every level of the organization (Web, internal development, database, sales, and customers).

The empirical study addresses the following research questions:

RQ1: (Sanity Check) Do the CCEA approach and the single population alternative significantly outperform random search?
RQ1.1: Does the single population GA outperform random search?
RQ1.2: Do the CCEAs outperform random search?

RQ2: (Effectiveness) How effective is the CCEA approach compared to the alternatives in terms of finding an earlier completion time?

RQ3: (Efficiency) Given the same number of evaluations, which algorithm finds the best-so-far solution soonest?

The population size in our implementation was set to 100 and the number of generations is listed in Table 1 for the three different configurations. For the CCEA, the fitness of an individual in either species depends on the results of its simulations with 6 individuals from the other species. Children compete with their parents, and, to select parents for reproduction, the algorithm randomly picks a number of parents and performs a tournament selection to identify parents for breeding. The pool of children is half the population size, i.e., 25. That is, the best 25 children had the chance to compete with their parents.

[1] The raw information of required effort for Project A was documented in 'person-hours' by the team that produced it. Since we do not know their mapping from hours to days, we leave the data in its raw form.

5 Empirical Study Results

In this section, we report results of the study described in Section 4.

5.1 Analysis of the Cooperative Co-evolutionary Progress

Results for all four projects and for the 3 CCEA configurations are plotted on Figures 4 and 5. In each sub-figure, the tick labels on the horizontal axis indicate the total number of internal generations that have been carried out, and also indicates the point at which the algorithm updated the population used for fitness computation. At each point on the horizontal axis, the entire population is depicted using a boxplot to give both a sense of the values obtained for completion time as the evolution progresses and the distribution of the fitness values in the population.

The fitness values of the entire population during the whole CCEA process are represented as boxplots. We can observe that the number of internal generations is the same for both populations within a specific sub-figure, as they fill equally spread vertical bands on the sub-figures. As can be seen from the sub-figures in the top rows in Figures 4 and 5, Config. I tends to find better solutions sooner than the other two configurations. On the second rows we can observe an noticeable interlacing of the co-evolutions between the two populations. For instance, as in Figure 4(c), the evolutionary process on each population takes 10 internal generations. The first 10 internal generations—plotted within the interval [1, 10] on the horizontal axis—record the evolutionary progress of TC. After the first round of evolution on TC, the solutions on WPO start evolving during the interval [11, 20]. On the third row of the sub-figures, it can be seen that the optimization of WPO produced more benefit—in terms of project completion time—than what done for TC. This can be noticed in Figure 4(e), where both species evolved for only one round (i.e., one external generation) and, in each round, they evolved for 100 generations (i.e., 100 internal generations). As indicated by the generation number on the horizontal axis, results from the TC species are plotted on the interval [1, 100], while results from the WPO species are plotted on the interval [101, 200]. As we can see from the completion time (vertical axis) the optimization of WPO leads to a noticeable improvement in the fitness function values obtained.

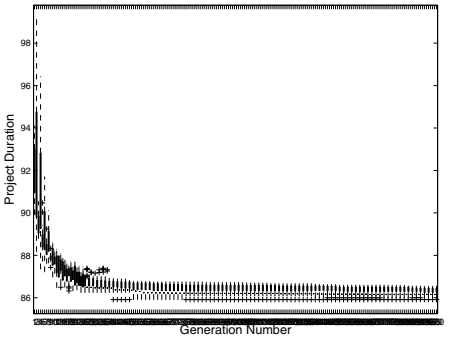
5.2 Results on Effectiveness

In this section we report the comparison of the effectiveness of three sets of CCEA configuration and the random search. Each algorithm was run 30 times on each of the 4 sets of project data to allow for statistical evaluation and comparison of the results.

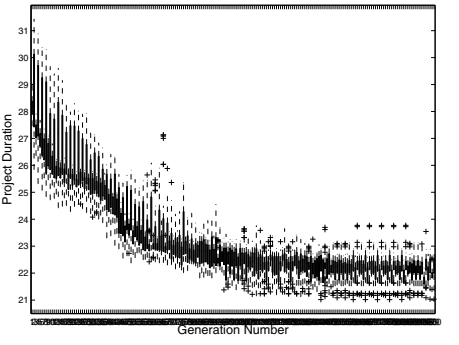
Figure 6 reports—for the various configurations—fitness values for the best individual solutions found by CCEA in the 30 runs.

As shown in the figures for Projects B, C, and D, in terms of the ability to effectively find the best solutions, CCEA performs better with Config. I and worse with Config. III. As explained in Section 3.3, Config. III is the single population evolutionary algorithm, while Config. I and II are bona fidè CCEAs. Therefore, our results provide evidence to support the claim that CCEAs outperform the single population evolutionary algorithm.

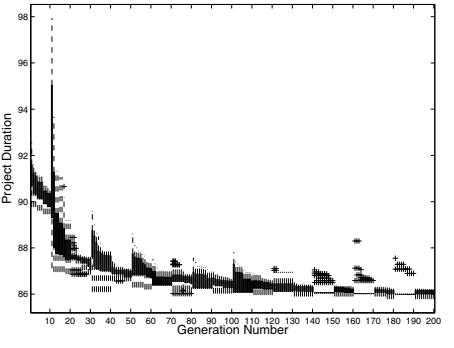
The random search generates twice the number of solutions of the CCEAs during the evolutionary process, and despite that, is clearly outperformed by the CCEAs in terms



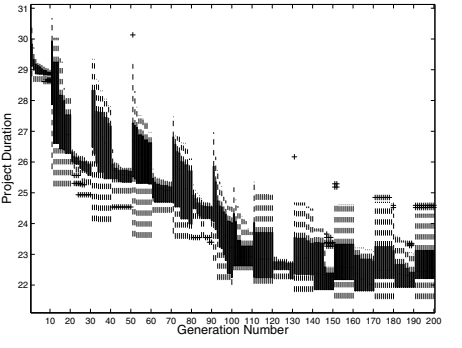
(a) Project A, Config. I



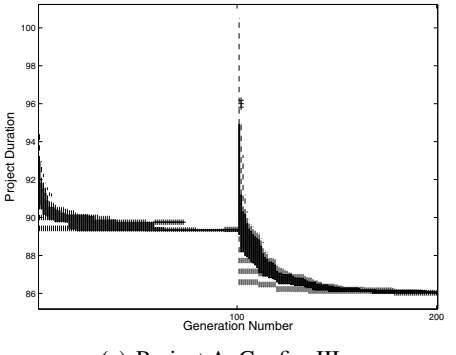
(b) Project B, Config. I



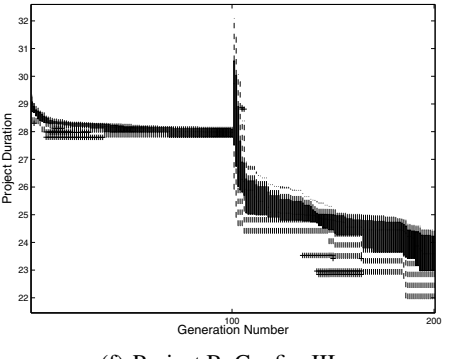
(c) Project A, Config. II



(d) Project B, Config. II

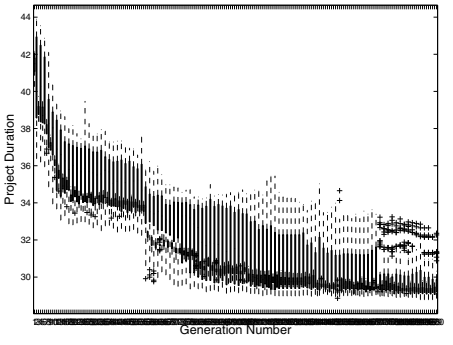


(e) Project A, Config. III

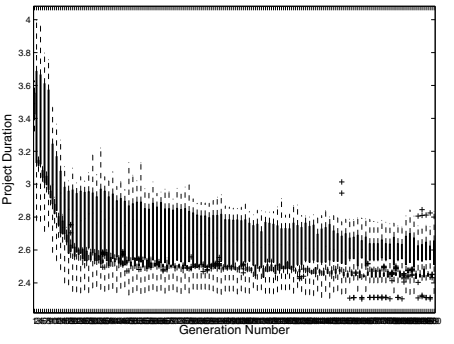


(f) Project B, Config. III

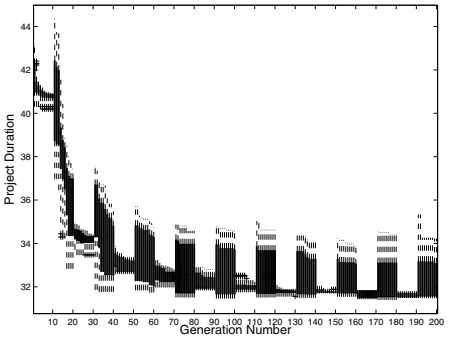
Fig. 4. Projects A and B: Boxplots of completion times for all solutions found by different CCEAs configurations



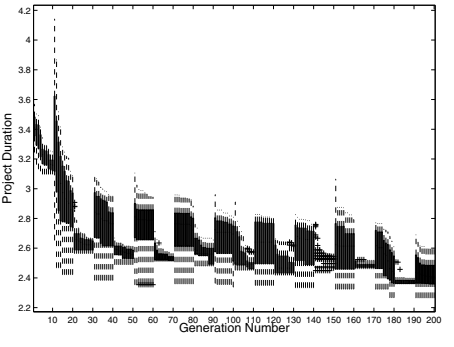
(a) Project C, Config. I



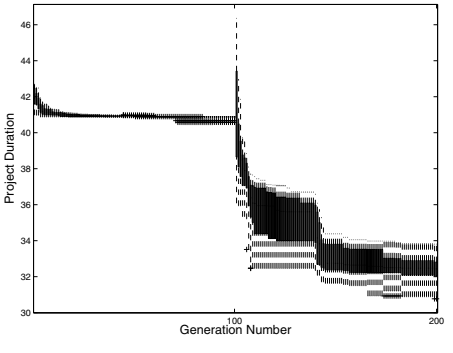
(b) Project D, Config. I



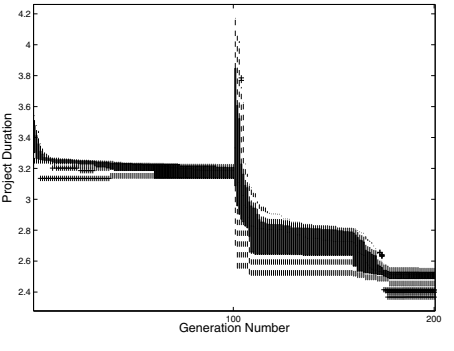
(c) Project C, Config. II



(d) Project D, Config. II



(e) Project C, Config. III



(f) Project D, Config. III

Fig. 5. Projects C and D: Boxplots of completion times for all solutions found by different CCEAs configurations

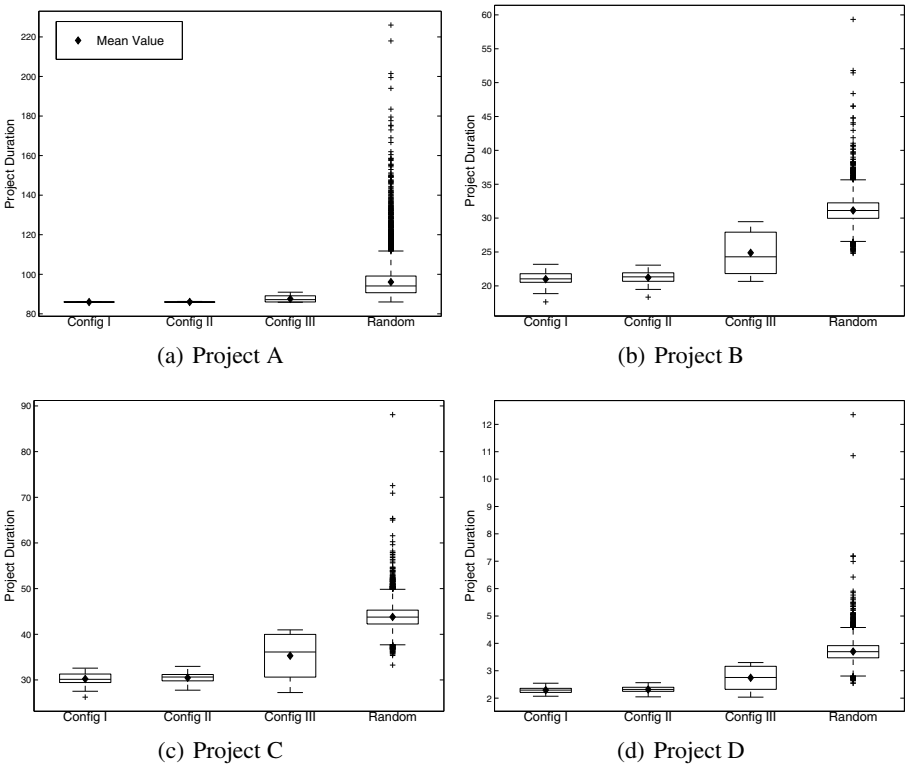


Fig. 6. Boxplots of all the best solutions found in 30 runs of the three CCEA configurations, and in random search runs

of fitness function quality. This observation is supported by a Wilcoxon Rank Sum Test (WRST) performed to calculate the statistical significance of the difference between the solutions produced by the different CCEAs configurations and Random. Since we are performing multiple comparisons on the same data set, *p-values* are corrected using the Holm's correction. The Wilcoxon Rank Sum Test (WRST) *p-values* reported in Table 3, as well as boxplots shown in Figure 6, indicate that all evolutionary algorithms perform significantly better than a random search, and that the best solutions found by the CCEAs (Config. I and II) perform significantly better than the single population evolutionary algorithm (Config. III).

For Project A, while all CCEAs perform significantly better than random search, the practical benefit in terms of lower project completion time achieved is not as evident as for the other projects. This is because in Project A there are no dependencies between WPs; the project is a conceptually simple, multiple application of a massive maintenance task (fixing Y2K problem repeatedly using a windowing approach). Since there are no dependencies, there is no delay introduced by the need for waiting on dependent WPs. For this reason, the WP scheduling and team construction have little impact on the overall completion time.

Table 3. Wilcoxon Rank Sum Test (unpaired) test adjusted p-values for the pairwise comparison of the three configurations

p-values for WRST	Projects			
	A	B	C	D
Config. I vs II	0.7229	0.1885	0.4481	0.2449
Config. I vs III	5.04E-08	3.00E-11	2.78E-07	2.19E-07
Config. II vs III	1.47E-07	8.86E-10	2.08E-06	1.28E-06
Config. I vs Random	3.97E-40	3.82E-40	3.83E-40	3.83E-40
Config. II vs Random	3.97E-40	3.82E-40	3.83E-40	3.83E-40
Config. III vs Random	2.70E-30	6.04E-37	3.13E-36	3.66E-36

In conclusion, the obtained results support the following two claims: (1) all three CCEAs were found to perform better than the random search, which means the CCEAs passed the 'sanity check' set by **RQ1**, and (2) **RQ2** is answered as the result of the WRST test that indicated the best solutions found by Config. I and II are significantly better than those found by Config. III. We conclude that there is evidence to suggest that co-evolution is effective to deal with software project staffing and scheduling.

5.3 Results on Efficiency

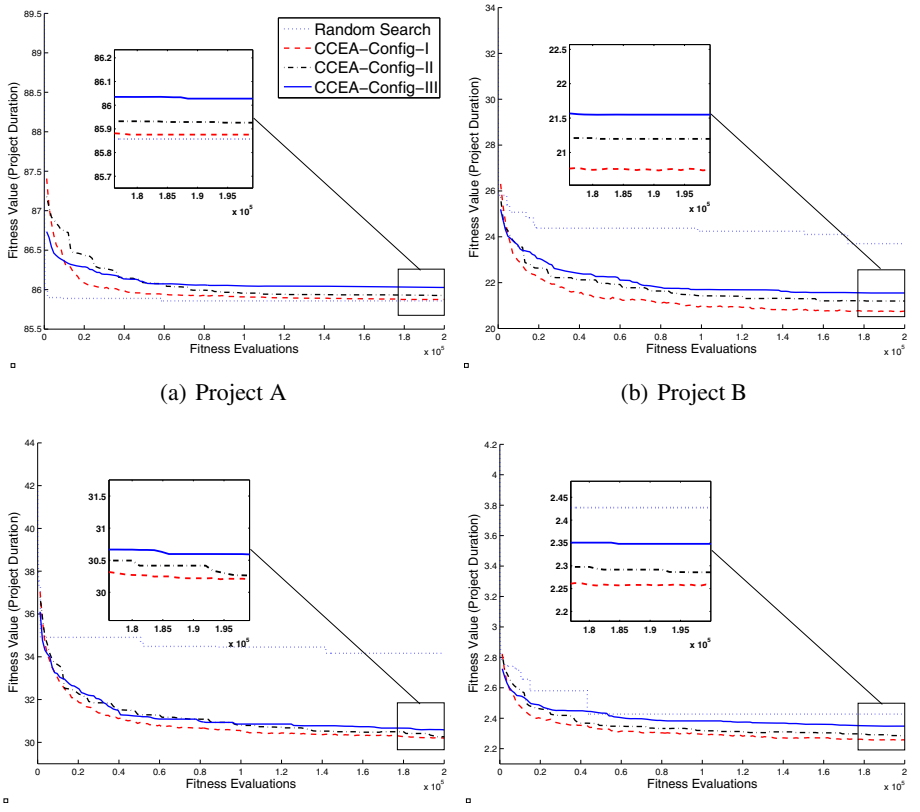
To answer **RQ3**, we extended the experiments with 30 runs of 3 CCEAs configurations until the solutions produced by all algorithms became stable, and, to allow a fair comparison, the random search was set to have the same number of fitness evaluations. The progress of the CCEAs and the random search in finding better solutions are plotted in Figure 7. The fitness values are averaged over 30 runs for CCEAs, while for the random search, the figure shows the best solutions found for the number of evaluations indicated on the horizontal axis.

As shown in Figures 7(b), 7(c), and 7(d), respectively for Projects B, C and D, in most cases, the CCEAs find better solutions than the non-cooperative algorithm. However, there is an exception found for Project A as shown in Figure 7(a), for which the CCEA does not outperform its rivals. We believe that this is due to the dependence-free nature of Project A (as mentioned before, it has no dependencies).

In conclusion, with regard to the efficiency of finding better solutions (**RQ3**), we find evidence that CCEAs outperform random search in general, and that the CCEA with more frequent communication between two populations (Config. I) performs better than the others (Config. II, III, and Random).

5.4 Threats to Validity

Construct validity threats may be due to the simplifications made when modeling the development/maintenance process through a simulation. In particular (i) we assumed communication overhead negligible and (ii) we did not consider developers' expertise. However, accounting for these variables was out of scope for this paper, as here the intent was to compare CCEA with non-co-evolutionary genetic algorithms.



(a) Project A (b) Project B

(c) Project C (d) Project D

Fig. 7. Efficiency Comparison of the Random Search and CCEAs

Threats to *internal validity* can be due, in this study, to the bias introduced in our results by the intrinsic randomness of GA and, of course, of the random approach. We mitigate such a threat by performing statistical tests on the results.

Threats to *conclusion validity* concern the relationship between treatment and outcome. Wherever possible, we use appropriate statistics—Wilcoxon test with Holm's correction in particular—to robustly test our conclusions.

Threats to *external validity* concern the generalization of our findings. We performed experiments on data from four industrial projects having different characteristics in terms of size, domain, and relationships among WPs. However, further studies are desirable to corroborate the obtained results.

6 Related Work

This section provides a brief overview of related work on search-based project planning.

Chang *et al.* were the first to publish on search-based project planning [10], with their introduction of the Software Project Management Net (SPMNet) approach for

project scheduling and resource allocation, which was evaluated on simulated project data. Aguilar-Ruiz *et al.* [2] also presented early results on evolutionary optimization for search-based project planning, once again evaluated on synthetically created software project data. Chicano and Alba [3,4] applied search algorithms to software projects to seek to find allocations that achieve earliest completion time. Alvarez-Valdes *et al.* [5] applied a scatter search approach to the problem of minimizing project completion duration.

Project management has recently [12] been the subject of a study of the human-competitiveness of SBSE, which found that optimization techniques are able to produce effective results in a shorter time than human decision makers. This work demonstrates that SBSE is a suitable approach to consider for project planning activities since it can find solutions that the human decision maker may otherwise miss. While the ultimate decision is likely to rest with the human decision maker, it is therefore important to find suitable SBSE techniques that can support this decision making activity.

Other authors have also worked on SBSE as a means of decision support for software engineering managers and decision makers in the planning stages of software engineering projects focusing on early lifecycle planning [8,11,18,19,23] as we do in the present paper, but also reaching forward to subsequent aspects of the software engineering lifecycle that also require planning, such as scheduling of bug fixing tasks [14,22]. Like our present work, some of this work has considered multiple objectives [4,15]. This is very natural in software project planning which is typified by many different concerns, each of which must be balanced against the others; and issue that is reported to be inherently as part of much work on SBSE [16]. However, no previous work has used co-evolution for project planning to blanch these different objectives.

SBSE can also be used as a way to analyze and understand Software Project Planning, yielding insight into planning issues, rather than seeking to necessarily provide a specific 'best' project plan [17]. For example SBSE has been used to study the effect of Brooks law [9] on project planning [20]. It has also been used to balance the competing concerns of risk and completion time [15]. Our work may be used in this way, since we can study the way the two populations evolve with respect to one another and the ways in which they are symbiotic. A thorough exploration of this possibility remains a topic for future work.

Di Penta *et al.* [13] compared the performance of different search-based optimization techniques, namely Genetic Algorithms, Simulated Annealing, and Hill Climbing to perform project planning on data from two industrial projects (Projects A and B also used in this study). The present paper can be thought of as an extension of the previous work of Di Penta *et al.*, because it uses the same representation and fitness function, while proposing and evaluating the use of completely unexplored optimization approach: co-evolutionary optimization.

7 Conclusions and Future Work

This paper proposes the use of Cooperative Co-Evolutionary Algorithms (CCEA) to solve software project planning and staffing problems. The co-evolutionary algorithm evolves two populations, one representing WP ordering in a queue (which determines

their assignment to teams), and the other representing developers distribution among teams.

We conducted an empirical study using data from four industrial software projects, aimed at comparing CCEA project planning and staffing with (i) random search and (ii) single population optimization using genetic algorithms, as previously proposed by Di Penta *et al.* [13]. Results of the empirical study show that CCEA is able to outperform random search and single population GA, in terms of effectiveness (i.e., best solutions proposed in terms of project completion time) and efficiency (i.e., a smaller number of evaluations required).

Future work aims at extending the study reported in this paper with further data sets and, above all, at considering a more sophisticated project model, which accounts for further factors not considered in this study, such as developers' skills and expertise, communication overhead models, and schedule robustness.

References

1. Adamopoulos, K., Harman, M., Hierons, R.M.: How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution. In: Deb, K., et al. (eds.) GECCO 2004. LNCS, vol. 3103, pp. 1338–1349. Springer, Heidelberg (2004)
2. Aguilar-Ruiz, J.S., Santos, J.C.R., Ramos, I.: Natural Evolutionary Coding: An Application to Estimating Software Development Projects. In: Proceedings of the 2002 Conference on Genetic and Evolutionary Computation (GECCO 2002), New York, USA, pp. 1–8 (2002)
3. Alba, E., Chicano, F.: Management of Software Projects with GAs. In: Proceedings of the 6th Metaheuristics International Conference (MIC 2005), pp. 13–18. Elsevier Science Inc., Austria (2005)
4. Alba, E., Chicano, F.: Software Project Management with GAs. Information Sciences 177(11), 2380–2401 (2007)
5. Alvarez-Valdes, R., Crespo, E., Tamarit, J.M., Villa, F.: A Scatter Search Algorithm for Project Scheduling under Partially Renewable Resources. Journal of Heuristics 12(1-2), 95–113 (2006)
6. Arcuri, A., Yao, X.: A Novel Co-evolutionary Approach to Automatic Software Bug Fixing. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2008), pp. 162–168. IEEE Computer Society, Hongkong (2008)
7. Arcuri, A., Yao, X.: Co-Evolutionary Automatic Programming for Software Development. Information Sciences (2010)
8. Barreto, A., de O. Barros, M., Werner, C.M.L.: Staffing a Software Project: a Constraint Satisfaction and Optimization-based Approach. Computers & Operations Research 35(10), 3073–3089 (2008)
9. Brooks Jr., F.P.: The Mythical Man Month: Essays on Software Engineering. Addison-Wesley Publishing Company, Reading (1975)
10. Chao, C., Komada, J., Liu, Q., Muteja, M., Alsalqan, Y., Chang, C.: An Application of Genetic Algorithms to Software Project Management. In: Proceedings of the 9th International Advanced Science and Technology, Chicago, Illinois, USA, pp. 247–252 (March 1993)
11. Cortellessa, V., Marinelli, F., Potena, P.: An Optimization Framework for Build-or-Buy Decisions in Software Architecture. Computers & Operations Research 35(10), 3090–3106 (2008)
12. de Souza, J.T., Maia, C.L., de Freitas, F.G., Coutinho, D.P.: The Human Competitiveness of Search Based Software Engineering. In: Second International Symposium on Search Based Software Engineering (SSBSE 2010), pp. 143–152 (2010)

13. Di Penta, M., Harman, M., Antoniol, G.: The Use of Search-Based Optimization Techniques to Schedule and Staff Software Projects: An Approach and An Empirical Study. Softw., Pract. Exper. 41(5), 495–519 (2011)
14. Fernando Netto, M.B., Alvim, A.: A Hybrid Heuristic Approach for Scheduling Bug Fix Tasks to Software. In: Proceedings of the 1st International Symposium on Search Based Software Engineering (SSBSE 2009). IEEE, UK (2009)
15. Gueorguiev, S., Harman, M., Antoniol, G.: Software Project Planning for Robustness and Completion Time in the Presence of Uncertainty using Multi Objective Search Based Software Engineering. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO 2009), pp. 1673–1680. ACM, Canada (2009)
16. Harman, M.: The Current State and Future of Search Based Software Engineering. In: Briand, L., Wolf, A. (eds.) Future of Software Engineering 2007, Los Alamitos, California, USA, pp. 342–357 (2007)
17. Harman, M.: The Relationship between Search Based Software Engineering and Predictive Modeling. In: 6th International Conference on Predictive Models in Software Engineering, Timisoara, Romania (2010)
18. Kapur, P., Ngo-The, A., Ruhe, G., Smith, A.: Optimized Staffing for Product Releases and Its Application at Chartwell Technology. Journal of Software Maintenance and Evolution: Research and Practice (Special Issue Search Based Software Engineering) 20(5), 365–386 (2008)
19. Kremmel, T., Kubalík, J., Biffl, S.: Software Project Portfolio Optimization with Advanced Multiobjective Evolutionary Algorithms. Applied Soft Computing 11(1), 1416–1426 (2011)
20. Penta, M.D., Harman, M., Antoniol, G., Qureshi, F.: The Effect of Communication Overhead on Software Maintenance Project Staffing: a Search-Based Approach. In: Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM 2007), pp. 315–324. IEEE, France (2007)
21. Potter, M.A., Couldrey, C.: A cooperative coevolutionary approach to partitional clustering. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 374–383. Springer, Heidelberg (2010)
22. Xiao, J., Afzal, W.: Search-based Resource Scheduling for Bug Fixing Tasks. In: Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE 2010), pp. 133–142. IEEE, Italy (2010)
23. Xiao, J., Osterweil, L.J., Wang, Q., Li, M.: Dynamic Resource Scheduling in Disruption-Prone Software Development Environments. In: Rosenblum, D.S., Taentzer, G. (eds.) FASE 2010. LNCS, vol. 6013, pp. 107–122. Springer, Heidelberg (2010)
24. Yang, Z., Tang, K., Yao, X.: Large Scale Evolutionary Optimization Using Cooperative Coevolution. Information Sciences 178(15), 2985–2999 (2008)