HPCLatAm 2013, pp. 13-24 (full paper)
Session: Evolutionary Computation & Scheduling
C. Garcia Garino and M. Printista (Eds.)
Mendoza, Argentina, July 29-30, 2013

Optimising Small-World Properties in VANETs with a Parallel Multi-Objective Coevolutionary Algorithm

Grégoire Danoy[1], Julien Schleich[1], Bernabé Dorronsoro[2], and Pascal Bouvry[1]

[1] Computer Science and Communications Research Unit
University of Luxembourg
{gregoire.danoy, julien.schleich, pascal.bouvry}@uni.lu,
[2] Laboratoire d'Informatique Fondamentale de Lille
University of Lille 1
bernabe.dorronsoro_diaz@inria.fr

Abstract. Cooperative coevolutionary evolutionary algorithms differ from standard evolutionary algorithms architecture in that the population is split into subpopulations, each of them optimising only a subvector of the global solution vector. All subpopulations cooperate by broadcasting their local partial solutions such that each subpopulation can evaluate complete solutions. Cooperative coevolution has recently been used in evolutionary multi-objective optimisation, but few works have exploited its parallelisation capabilities or tackled real-world problems. This article proposes to apply for the first time a state-of-the-art parallel asynchronous cooperative coevolutionary variant of the non-dominated sorting genetic algorithm II (NSGA-II), named CCNSGA-II, on the injection network problem in vehicular ad hoc networks (VANETs). This multi-objective optimisation problem, consists in finding the minimal set of nodes with backend connectivity, referred to as injection points, to constitute a fully connected overlay that will optimise the small-world properties of the resulting network. Recently, the well-known NSGA-II algorithm was used to tackle this problem on realistic instances in the city-centre of Luxembourg. In this work we compare the performance of the CCNSGA-II to the original NSGA-II in terms of both quality of the obtained Pareto front approximations and execution time speedup.

1 Introduction

Real-world optimisation problems are typically hard and multi-objective by nature, since different conflicting criteria have to be considered. This means that in such problems improving one objective will imply decreasing (some of) the others. Contrary to single-objective optimisation which aims to a unique solution, multi-objective optimisation consists in finding a set of so-called non-dominated solutions in the objective space, referred to as Pareto-front. A solution is said to dominate another one if it is better on one objective and similar or better on the

other objectives. A multi-objective algorithm then aims to find a limited set of non-dominated solutions as close as possible to the optimal Pareto-front, both in terms of diversity and convergence.

Since many years, Evolutionary algorithms (EAs) have proven to be an efficient approach for multi-objective optimisation [1]. However standard multi-objective EAs have shown some limits when dealing with large and complex problems, which motivated research in faster and more accurate methods. One promising approach is cooperative coevolution as introduced originally by Potter for single-objective optimisation [12], in which the solution vector is decomposed and each subset of solution is evolved in a separated subpopulation. These cooperate by exchanging their local representative(s) in order to create a complete solution to be evaluated on the global problem.

Some recent works have demonstrated the advantages of cooperative coevolution in the multi-objective context [5, 14], most of them focused on improving the solutions quality. As originally stated by Potter, cooperative coevolution has some good potential for parallelism, but only few works have studied this capability. Most lately, Nielsen *et al.* proposed a new parallel asynchronous cooperative coevolutionary non-dominated sorting genetic algorithm II (NSGA-II) [10] that demonstrated promising speedup capabilities but limited their study to well-known multi-objective benchmark functions.

In this work we thus propose to apply for the first time this state-of-the-art algorithm on a real-world problem, i.e., the injection network problem in vehicular ad hoc networks (VANETs). This topology control problem consists in finding the minimum set of vehicles, called injections points, chosen to provide backend connectivity and compose a fully-connected overlay network such that small-world properties of the resulting network are optimised. Experiments are conducted on realistic VANET networks snapshots from Luxembourg city, generated with the VehILux mobility model [11].

The remainder of this article is organised as follows. The next section presents a brief state-of-the-art on cooperative coevolutionary multi-objective EAs. Then the NSGA-II algorithm and its asynchronous cooperative coevolutionary variant (CCNSGA-II) are presented in detail in section 3. The injection network problem is described in section 4. Section 5 then provides the experimental setup and results analysis in terms of execution time speedup and solution quality. Finally our conclusions and perspectives and given in section 6.

2 Related Work

Cooperative coevolutionary evolutionary algorithms (CCEAs) mainly differ from standard EAs by their fitness evaluation that requires interactions individuals from other so-called subpopulations. In the original cooperative coevolution framework proposed by Potter *et al.* in [12], the decision vector is split such that each decision variable is evolved in different subpopulations. In order to evaluate their partial solutions, each subpopulation exchanges periodically some representative (e.g. best) with all other subpopulations. This coevolutionary framework

was initially proposed for single-objective optimisation but more recently several multi-objective versions proved to be efficient [1]. The following proposes a brief survey of cooperative coevolutionary MOEAs, a more comprehensive one can be found in [4].

The first CCMOEA was proposed in [8] as an extension of the Genetic Symbiosis Algorithm (GSA) for multi-objective optimisation problems. The multi-objective GSA (MOGSA) differs from the standard GSA with a second symbiotic parameters, which represents the interactions of the objective functions. One main drawback of this algorithm is that it requires knowledge of the search space, which highly reduces its application possibilities.

Then, Keerativuttitumrong *et al.* proposed the Multiobjective Co-operative Co-evolutionary Genetic Algorithm (MOCCGA) in [6]. It combines Fonseca and Flemings multi-objective GA (MOGA) and Potter's Co-operative Co-evolutionary Genetic Algorithm (CCGA). Each subpopulation evolves using MOGA and assigns a fitness to its individuals based on their rank in the subpopulation local Pareto front. However this local Pareto optimality perception is a limiting factor for the performance of MOCCGA. A parallel implementation of MOCCGA was empirically validated using 1, 2, 4 and 8 cores, but limited to well-known benchmark functions (ZDT [16]).

MOCCGA was extended in [7], with other MOEAs and a fixed size archive that stores the non-dominated solutions. Another variant was introduced by Tan *et al.* [14] proposed, in addition to adding an archive, a novel adaptive niching mechanism. A parallel version was also experimented but still limited to the ZDT benchmark. Finally, [5] presented a non-dominated sorting cooperative coevolutionary algorithm (NSCCGA), which is essentially the coevolutionary extension of NSGA-II.

Most lately, Dorronsoro *et al.* proposed a parallel synchronous CCMOEA framework with three different MOEAs [4]: NSGA-II, Strength Pareto Evolutionary Algorithm (SPEA2) [17] and Multi-objective Cellular Genetic Algorithm (MOCell) [9]. They demonstrated that super-linear speedup is possible on a scheduling problem, compared to the original MOEAs. Finally, a new variant with asynchronous communications between the subpopulations was proposed in [10] which further improved the speedup without degrading the solutions quality on standard benchmarks (i.e., DTLZ [3] and ZDT).

This article proposes to apply for the first time this parallel asynchronous CCMOEA with NSGA-II on a real-world problem, i.e., the optimisation of small-world properties in VANETs.

3 Algorithms

The following subsections present the two multi-objective algorithms considered in this work, the well-known Non-dominated Sorting Genetic Algorithm II (NSGA-II) and its cooperative coevolutionary variant CCNSGA-II.

3.1 Non-dominated Sorting Genetic Algorithm (NSGA-II)

The NSGA-II [2] algorithm is the reference algorithm in multi-objective optimisation. A pseudocode is given in Algorithm 1. NSGA-II does not implement an external archive of non-dominated solutions, but the population itself keeps the best non-dominated solutions found so far. The algorithm starts by generating an initial random population and evaluating it (lines 2 and 3). Then, it enters in the main loop to evolve the population. It starts by generating a second population of the same size as the main one. It is built by iteratively selecting two parents (line 6) by binary tournament based on dominance and crowding distance (in the case the two selected solutions are non-dominated), recombining them (two-point crossover in our case) to generate two new solutions (line 7), which are mutated in line 8 (here bit flip mutation) and added to the offspring population (line 9). The number of times this cycle (lines 5 to 10) is repeated is the population size divided by two, thus generating the new population with the same size as the main one. This new population is then evaluated (line 11), and merged with the main population (line 12). Now, the algorithm must discard half of the solutions from the merged population to generate the population for the next generation. This is done by selecting the best solutions according to ranking and crowding, in that order. Concretely, ranking consists on ordering solutions according to the dominance level into different fronts (line 13). The first front is composed by the non-dominated solutions in the merged population. Then, these solutions in the first front are removed from the merged population, and the non-dominated ones of the remaining solutions compose the second front. The algorithm iterates like this until all solutions are classified. To build the new population for the next generation, the algorithm adds those solutions in the first fronts until the population is full or adding a front would suppose exceeding the population size (line 14). In the latter case (lines 15 to 17), the best solutions from the latter front according to crowding distance (i.e., those solutions that are more isolated in the front) are selected to complete the population. The process is repeated until the termination condition is met (lines 4 to 18).

```
1: //Algorithm parameters in 'nsga'
2: InitialisePopulation(nsga.pop);
3: EvaluatePopulation(nsga.pop);
4: while ! StopCondition() do
5:    for index ← 1 to cga.popSize/2 do
6:       parents←SelectParents(nsga.pop);
7:       children←Crossover(nsga.Pc,parents);
8:       children←Mutate(nsga.Pm,children);
9:       offspringPop←Add(children);
10:   end for
11:   EvaluatePopulation(offspringPop);
12:   union←Merge(nsga.pop, offspringPop);
13:   fronts←SortFronts(union);
14:   (Pop', lastFront)←GetBestCompleteFronts(fronts);
15:   if size(nextPop) < nsga.popsize then
16:      Pop'←BestAccToCrowding(lastFront,nsga.popsize-size(Pop'));
17:   end if
18: end while
```

Algorithm 1: Pseudocode for NSGA-II

3.2 Asynchronous Cooperative Coevolutionary NSGA-II

As previously mentioned, cooperative coevolution splits the solution vector and evolves each subset of the solution using a genetic algorithm, in our case NSGA-II, in so-called subpopulations. In the single-objective case, each subpopulation then broadcasts its representatives to all the other subpopulations after each generations following a selection scheme (e.g., best individual). This fully connected broadcast of representatives enables each subpopulation to assemble and evaluate the resulting global objective function which is essential for the local genetic algorithm.

To apply the coevolutionary paradigm in the multi-objective context, several changes to the single-objective framework must be operated. The CCNSGA-II is based on the asynchronous cooperative coevolutionary framework which pseudo-code is given in Algorithm 2 and described in the following. A first difference is the merge of the local Pareto-front from each sub-population to generate an archive of best combined Pareto-fronts, as can be seen in line 16. This merging process of the solutions sets is done by choosing one of the sets and adding to it all the other solutions. If the resulting approximation set exceeds the archive size, the crowding policy is used to remove solutions based on the distance to surrounding individuals belonging to the same rank.

A second difference induced by the multi-objective design lies in the construction of complete solutions for evaluation. As previously mentioned, in the single-objective case a partial solution from one subpopluation is evaluated by composing a complete solution with the best partial solutions received from all the other subpopulations. Whereas, in the multi-objective case, there will be most of the time more than a single best solution, i.e., a set of non-dominated solutions. In the CCNSGA-II, every subpopulation shares a number N_s of solutions randomly chosen from the non-dominated ones found so far. An example of how one sub-population, P_1, shares four of its best solutions (i.e., $N_s = 4$) with the other two populations is presented in Fig. 1. In case the local Pareto-front

```
1: t ← 0
2: {⫠ means parallel run}
3: ⫠ i ∈ [1, I] :: setup( P^0 , i ) {Initialise every subpopulation}
4: sync() {Synchronisation point}
5: {∀ means sequential run}
6: ∀ i ∈ [1, I] :: broadcast( P^0 , i ) {Send random local partial solutions to all subpopulations}
7: ⫠ i ∈ [1, I] :: evaluate( P^0 , i ) {Evaluate solutions in every subpopulation}
8: sync()
9: ⫠ i ∈ [1, I] :: {
10: while not stoppingCondition( ) do
11:     generation( P^t , i ) {Perform one generation to evolve the population}
12:     broadcast( P^t , i ) {Share best local partial solutions in every subpopulation}
13:
14:     t ← t + 1 {Increase generations counter}
15: end while
    }
16: mergeParetoFronts( ) {Merge all subpopulations Pareto fronts into a single one}
```

Algorithm 2: Asynchronous CCNSGA-II framework

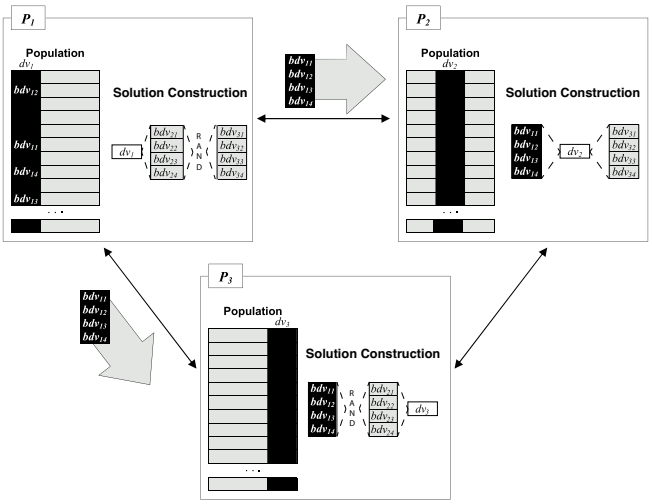

Fig. 1. In the CCMOEA, every population (for example, $P1$) shares with the other coevolving populations (P_2 and P_3) its four best partial solutions (bdv_{11} to bdv_{14}). The partial solutions are evaluated by building complete solutions with random partial solutions of the other two subpopulations (bdv_{2X} and bdv_{3Y})

contains less than N_s non-dominated solutions, randomly chosen individuals are taken from the rest of the population to complete the set of N_s solutions.

In this asynchronous variant, contrary to the original synchronous model, there is no synchronisation point before the broadcasting of the solutions (i.e., synchronisation would be inserted in line 12). Subpopulations keep evolving independently even when the other subpopulations are still busy with the current generation. This means that the subpopulations will not share the total number of available objective function evaluations equally between them. A fast executing subpopulation might perform more evaluations and thus 'steal' evaluations from the slower subpopulations. Even if all subpopulations do 'consume' the same amount of total function evaluations, the asynchronous version should remove delays and allow the algorithm to finish faster. Another possible consequence of the asynchronicity is that a subpopulation may start evaluating its individuals before the other subpopulations have transmitted their own individuals, i.e., old individuals are used. However this effect was shown to have no statistical significant impact on the results quality compared to the synchronous model [10].

4 Problem Description

The injection network problem considered in this work was originally introduced in [13]. Provided a snapshot of a VANET, the objective is to determine the best set of vehicles to join the overlay network in order to unpartition the cor-

responding network graph and maximise its small-world properties. In the first subsection the injection network problem is defined together with the small-world metrics used. The second subsection defines the corresponding multi-objective optimisation problem.

4.1 Injection Networks

This problem considers hybrid VANETs where each vehicle can potentially have both vehicle-to-vehicle and vehicle-to-infrastructure (e.g., using Wi-Fi hotspots) communications. Nodes elected as injection points (i.e., nodes connected to the infrastructure) form a fully connected overlay network, that aims at increasing the connectivity and robustness of the VANET. Injection points respectively permit to efficiently disseminate information from distant and potentially disconnected nodes and prevent costly bandwidth overuse with redundant information. An example network is presented in Figure 2.

In this problem, we consider small-world properties as indicators for the good set of rules to choose injection points. Small-World networks [15] are a class of graphs that combines the advantages of both regular and random networks with respectively a high clustering coefficient (CC) and a low average path length (APL). The APL is defined as the average of the shortest path length between any two nodes in a graph $G = (V, E)$, that is $APL = \frac{1}{n(n-1)} \sum_{i,j} d(v_i, v_j)$ with $d(v_i, v_j)$ the shortest distance between nodes v_i, $v_j \in V$. It thus indicates the degree of separation between the nodes in the graph. The local CC of node v with k_v neighbours is $CC_v = \frac{|E(\Gamma_v)|}{k_v(k_v-1)}$ where $|E(\Gamma_v)|$ is the number of links in the relational neighbourhood of v and $k_v(k_v - 1)$ is the number of possible links in the relational neighbourhood of v. The global clustering coefficient is the average of all local CC in the network, denoted as $CC = \frac{1}{n} \sum_v CC_v$. The CC measures to which extent strongly interconnected groups of nodes exist in the network, i.e., groups with many edges connecting nodes belonging to the group, but very few edges leading out of the group.

We here consider Watts original definition of the small world phenomenon in networks with $APL \approx APL_{random}$ and $CC \gg CC_{random}$, where APL_{random} and CC_{random} are, respectively, the APL and CC of random graphs with similar number of nodes and average node degree k. In addition, the number of chosen injection points has to be minimised as they may induce additional communication costs.

4.2 Optimisation Problem

The proposed optimization problem can be formalized as follows. The solution to this problem is a binary vector s of size n (number of nodes in the network), $s[1..n]$ where $s[i] = 1$ if node v_i is an injection point, and $s[i] = 0$ if v_i is not an injection point. The decision space is thus of size 2^n.

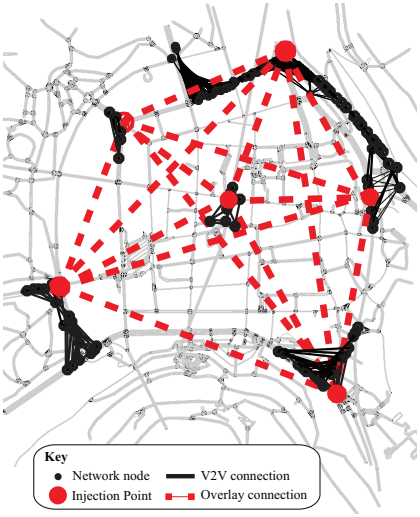

Fig. 2. Network with 248 nodes including 6 injections points composing the overlay network in Luxembourg city.

This problem is a three objectives one, defined as:

$$f(s) = \begin{cases} min\,\{inj\} \\ max\,\{cc\}\ ; \\ min\,\{apl_{diff}\} \end{cases} \quad \text{s. t. } component = 1 \tag{1}$$

where inj is the number of chosen injection points, cc is the average clustering coefficient of the resulting network, and apl_{diff} is the absolute difference between the APL of the resulting network and the APL of the equivalent random graph: $apl_{diff} = |apl - apl_{random}|$. For each overlay network instance evaluated in this work, the apl_{random} and cc_{random} is obtained by averaging the APL and CC of 30 corresponding random graphs using Watts random rewiring procedure with probability $p = 1$ [15]. Each random graph is created based on the number of devices and average network degree. Since the initial objective is to unpartition the network, a constraint is set on the number of connected components in the created network, i.e., *component* must be equal to 1.

5 Experiments

This section first describes the methodology we followed for our experiments, i.e., the algorithms and problem instances parameters. In the second subsection the obtained results considering both solutions quality and speedup are presented and analysed.

5.1 Experimental setup

The configuration of the NSGA-II algorithm used in the subpopulations is the same as for the 'standard' NSGA-II, except the length of the decision vector. Indeed, for the NSGA-II, the length is equal to the total number of nodes in the problem instance, whereas for the CCNSGA-II it is divided by the number of subpopulations.

The parameters of the algorithms are presented in Table 1. The configuration of the NSGA-II is the one originally suggested by the authors [2]. However, some parameters were adapted because a binary representation was used, in which each gene (i.e., bit) represents one car. Genes set to 1 mean that the corresponding cars act as injection points, while a 0 value indicates the contrary The two-point crossover operator (DPX) with probability $p_c = 0.9$ and the bit-flip mutation operator with probability $p_m = 1/number\_of\_variables$ were used. In two-point recombination, two crossover positions are selected uniformly at random in both parents, and the values between these points are exchanged to generate two new offspring solutions. The bit-flip mutation is to change a 1 into a 0, or vice-versa. The algorithm evolves until $50,000$ fitness function evaluations are performed, and 30 independent runs were executed for every problem instance.

The CCNSGA-II uses 4 subpopulations, each of them is run in a separate thread running on a different core of the same multi-processor machine. The population size for the NSGA-II is 100, and every subpopulation also has 100 individuals in the studied CCNSGA-II. Islands are exchanging 20 randomly chosen local non-dominated solutions, and to evaluate a given solution, it is built with random sub-solutions from those shared by the other subpopulations, as proposed in [4].

For the problem instances, we have used snapshots from realistic VANET scenarios in the city centre of Luxembourg, simulated using the VehILux mobility model [11]. VehILux accurately reproduces the vehicular mobility in Luxembourg

Table 1. Algorithms configuration.

Numb. of subpop.*	4
Cores used	4 (1 for NSGA-II)
Number of threads	1 per subpopulation
Population size	100
Final archive size	100, from all subpops.
Migration policy *	20 random
Max. evaluations	$50,000$
Pop. initialisation	Random
Selection	Binary tournament
Recombination	DPX
Probability	0.9
Mutation	Bit Flip
Probability	$\frac{1}{number\_of\_variables}$
Independent runs	30

* Not applicable for NSGA-II

Table 2. Network instances

	Surface	$0.6\ km^2$		
	Coverage radius	100 m		
6 a.m.	Network Number	21900	22200	22500
	Number of Nodes	40	62	60
	Partitions	10	8	6
	Solution space size	1^{12}	4.61^{18}	1.15^{18}
7 a.m.	Network Number	25500	25800	26099
	Number of Nodes	223	248	301
	Partitions	10	6	7
	Solution space size	1.34^{67}	4.52^{74}	4.07^{90}

by exploiting both realistic road network topology (OpenStreetMaps) and real traffic counting data from the Luxembourg Ministry of Transport. The 6 studied networks represent snapshots of a simulated area of 0.6 km^2, the first three small snapshots are taken between 6:00 a.m. and 6:15 a.m. and the three large ones between 7:00 a.m. and 7:15 a.m. The properties of the instances are shown in Table 2.

5.2 Experimental results

This section presents the results obtained in our experiments. These were run on the HPC facility of the University of Luxembourg. The nodes used are HP Proliant BL2x220c G6 (10U) with 2 Intel L5640 CPUs having 6 cores each at 2.26 GHz.

The first part focuses on the analysis of the speedup obtained and the second part analyses the solution quality using the following two well-known quality measures to assess the quality (solutions diversity and convergence) of the found solutions by the two algorithms: unary additive epsilon ($I^1_{\varepsilon+}$) and spread (I_S).

Speedup. The first objective of this work is to study the benefit of the parallel asynchronous CCNSGA-II in terms of computational time speedup compared to the standard NSGA-II. Figure 3 presents the average execution times in seconds for the six problem instances and the corresponding speedup factor. The execution times increase drastically together with the problems instance sizes, from 121.18 seconds for the fastest with CCNSGA-II on the smallest instance (21900), to more than 107 hours with NSGA-II on the largest instance (26099). This very large computational time justifies the search for efficient parallel optimisation approaches. The average speedup obtained is 4.55 with a low standard deviation of 0.2. Indeed the obtained speedup is quite constant and always super-linear (i.e., above 4), the worst one being obtained on the 22500 instance with 4.20 and the best one on the 25500 with 4.76, which confirms the promising results obtained on test functions in [10] despite being slightly lower.

Solutions quality. We here propose to analyse the quality of the Pareto fronts obtained with the two algorithms, to verify that the speedup obtained with the

Instance	NSGA-II	CCNSGA-II	Speedup
21900	569.62	121.18	4.70
22200	2159.40	479.35	4.50
22500	1892.97	450.23	4.20
25500	141387.52	29676.34	4.76
25800	192580.75	42622.23	4.52
26099	386924.15	83420.32	4.64

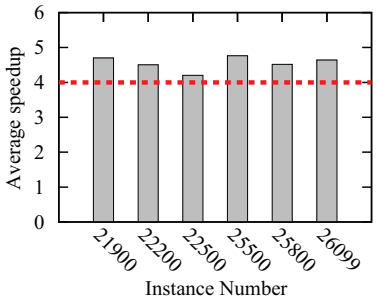

Fig. 3. Execution times and speedup numbers (left) and speedup plots (right).

Table 3. Quality measures for the NSGA-II and asynchronous CCNSGA-II

	SPREAD			EPSILON		
Instance	**NSGA-II**	**CCNSGA-II**	Conf.	**NSGA-II**	**CCNSGA-II**	Conf.
21900	$6.55e-01_{4.5e-02}$	$7.32e-01_{7.9e-02}$	▲	$7.00e+00_{0.0e+00}$	$1.67e+00_{2.1e-08}$	▽
22200	$6.85e-01_{4.6e-02}$	$7.70e-01_{4.7e-02}$	▲	$3.00e+00_{0.0e+00}$	$1.40e+00_{7.0e-02}$	▽
22500	$7.25e-01_{3.8e-02}$	$7.44e-01_{6.7e-02}$	−	$4.00e+00_{0.0e+00}$	$1.90e+00_{2.9e-02}$	▽
25500	$8.56e-01_{1.1e-01}$	$7.41e-01_{4.1e-02}$	▽	$1.57e+01_{4.3e+00}$	$2.93e+00_{9.0e-01}$	▽
25800	$8.36e-01_{7.2e-02}$	$7.49e-01_{7.0e-02}$	▽	$8.87e+00_{4.1e+00}$	$3.14e+00_{1.2e+00}$	▽
26099	$7.31e-01_{1.1e-01}$	$7.86e-01_{5.2e-02}$	▲	$1.56e+01_{6.7e+00}$	$4.00e+00_{1.4e+00}$	▽

parallel coevolutionary algorithm does not come to the price of lower solution quality.

Table 3 presents the results we obtained (as mean and standard deviation) with NSGA-II and CCNSGA-II for the six considered problem instances according to the spread (Δ), and unary additive epsilon ($I^1_{\varepsilon+}$) quality metrics. The best obtained results are shadowed in with dark grey colour. Statistical confidence in the results is assessed using the Wilcoxon unpaired signed-ranks test. It is represented in the Table by ▲ if NSGA-II is statistically better, by ▽ if statistically worse and by − if no significant difference was found.

Regarding the diversity of solutions measured by the SPREAD, the NSGA-II is statistically better for half of the instances (i.e., 21900, 22200 and 26099), while the CCNSGA-II is better for two instances and finally no statistical difference is found on the 25500 instance. The difference in terms of convergence is very significant since CCNSGA-II outperforms NSGA-II in all instances with statistical confidence. These results demonstrate that in average the CCNSGA-II permits to improve the quality of solutions of the NSGA-II.

6 Conclusion

This article has proposed to apply for the first time a parallel asynchronous cooperative coevolutionary NSGA-II (CCNSGA-II) on a real-world problem, i.e., the injection network problem in VANETs. Its performance in terms of execution time and solution quality has been compared to the standard NSGA-II. Experimental results have demonstrated that super-linear speedup has been obtained on all problem instances. In addition, the convergence of the obtained Pareto fronts could be improved with statistical confidence for all cases as well as their diversity for some of them.

Future works will focus on studying the possible additional speedup with higher numbers of subpopulations and on using other state-of-the-art MOEAs like SPEA2 and MOCell on the same topology control problem in VANETs.

Acknowledgements

B. Dorronsoro acknowledges the support offered by the National Research Fund, Luxembourg, AFR contract no 4017742.

References

1. C. A. Coello Coello, G. B. Lamont, and D. A. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2007. Second edition.
2. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.
3. K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Congress on Evolutionary Computation (CEC)*, pages 825–830, 2002.
4. B. Dorronsoro, G. Danoy, A. J. Nebro, and P. Bouvry. Achieving super-linear performance in parallel multi-objective evolutionary algorithms by means of cooperative coevolution. *Computers & Operations Research*, 40(6):1552 – 1563, 2013.
5. A. W. Iorio and X. Li. A cooperative coevolutionary multiobjective algorithm using non-dominated sorting. In *GECCO (1)*, pages 537–548, 2004.
6. N. Keerativuttitumrong, N. Chaiyaratana, and V. Varavithya. Multi-objective co-operative co-evolutionary genetic algorithm. In *International Conference on Parallel Problem Solving From Nature PPSN*, volume 2439 of *Lecture Notes in Computer Science (LNCS)*, pages 288–297. Springer-Verlag, 2002.
7. K. Maneeratana, K. Boonlong, and N. Chaiyaratana. Multi-objective optimisation by co-operative co-evolution. In *PPSN*, pages 772–781, 2004.
8. J. Mao, K. Hirasawa, J. Hu, , and J. Murata. Genetic symbiosis algorithm for multiobjective optimization problem. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 267–274, San Francisco, California, 2001.
9. A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. MOCell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24:726–746, 2009.
10. S.S. Nielsen, B. Dorronsoro, G. Danoy, and P. Bouvry. Novel efficient asynchronous cooperative co-evolutionary multi-objective algorithms. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–7, 2012.
11. Y. Pigné, G. Danoy, and P. Bouvry. A vehicular mobility model based on real traffic counting data. In *Proc. 3rd International Workshop on Communication Technologies for Vehicles (Nets4Cars 2011)*, volume 6596, pages 131–142. Springer, LNCS, 2011.
12. M.A. Potter and K. De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature (PPSN III)*, pages 249–257. Springer, 1994.
13. J. Schleich, G. Danoy, B. Dorronsoro, and P. Bouvry. An overlay approach for optimising small-world properties in vanets. In AnnaI. Esparcia-Alczar, editor, *Applications of Evolutionary Computation*, volume 7835 of *Lecture Notes in Computer Science*, pages 32–41. Springer Berlin Heidelberg, 2013.
14. K. C. Tan, Y. J. Yang, and C. K. Goh. A distributed cooperative coevolutionary algorithm for multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 10(5):527–549, 2006.
15. D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, 1998.
16. E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.
17. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 2001.