A parallel micro evolutionary algorithm for taxi sharing optimization

Renzo Massobrio, Gabriel Fagúndez and Sergio Nesmachnow
Centro de Cálculo, Facultad de Ingeniería, Universidad de la República
Herrera y Reissig 565, 11300 Montevideo, Uruguay
Email: {renzom, gabrielf, sergion}@fing.edu.uy

***Abstract*—This article presents the application of a parallel micro evolutionary algorithm to the problem of distributing passengers traveling from the same origin to different destinations in several taxis, with the goal of minimizing the total cost of the trips. The proposed method is designed to provide an accurate and efficient way to solve the problem. The experimental analysis compares the solutions found using the proposed algorithm versus those computed using a sequential evolutionary algorithm and an intuitive greedy heuristic. The results show that the parallel evolutionary algorithm is able to efficiently reach significant improvements in the total cost, outperforming the greedy heuristic in up to 36.1% (18.2% on average), and the sequential evolutionary algorithm in up to 8.5% (4.3% on average).**

I. Introduction

Car pooling is the concept of sharing car journeys, and it has gained massive public attention in recent years [1]. Both economical and environmental benefits (at individual and collective levels) are obtained by sharing car trips, as it minimizes travel expenses as well as the amount of vehicles on the streets, thus reducing pollution. This contributes to minimize the impact of transportation in the environment, which is a major concern nowadays, especially for big cities [2] [3]. Additionally, less traffic leads to fewer traffic jams, resulting in a more fluent, thus more efficient, trip. From an economical perspective, sharing trips between multiple passengers significantly reduces the transportation costs.

The aforementioned benefits have led to several initiatives to attend the public concern on this topic. Exclusive car-pool lanes, campaigns to promote car sharing when commuting to work, and a plethora of mobile applications to find car pooling mates [4], are some of the many examples that illustrate the importance of this subject.

Taxis are a fast and reliable mean of transportation. However, they rarely run at full capacity and could therefore benefit from the car pooling idea. There are some web platforms that provide solutions for ride-sharing scheduling, and some research works on different variants of the taxi-pooling problem. The taxi sharing problem is NP-hard [5]. Thus, heuristics and metaheuristics [6] are needed to find high-quality solutions in reasonable execution times for realistic problem instances.

In this line of work, this article introduces a parallel micro evolutionary algorithm to solve the one origin to multiple destinations variant of the taxi sharing problem. The experimental evaluation, performed over real-world scenarios, demonstrates that the proposed method is an accurate and efficient tool to solve the problem, which can be easily integrated in on-line (web and/or mobile) applications.

The article is organized as follows. Section II introduces the taxi sharing problem and reviews related work. Section III introduces evolutionary algorithms (EAs), and the proposed micro EA to solve the problem. Section IV reports the experimental evaluation, including a comparison against a sequential EA from [12] and a greedy heuristic to solve the problem. Finally, Section V formulates the conclusions and the main lines for future work.

II. The taxi sharing problem

This section introduces the taxi sharing problem and reviews related works on the topic.

A. Problem model and formulation

The problem models the reality of a group of people (the *passengers*) willing to share a taxi from the same origin to different destinations. Passengers are interested in knowing the appropriate number of taxis needed and how to visit their destinations in order to minimize the total cost.

Passenger distribution is restricted to the maximum number of people allowed by safety regulations to travel in each taxi. The transportation costs model realistic data in a fluid traffic scenario (including a model for delays and traffic congestion is proposed as future work). The cost for each taxi includes the cost to hire the taxi (*minimum fare*), and the cost for traveling from the origin to the final destination; additional costs related to baggage, tips, or waiting times are not considered.

The mathematical formulation of the taxi sharing problem considers the following elements:

- A set of passengers $P = \{p_1, p_2, \ldots, p_N\}$; travelling from the same origin point O to a set of (potentially different) destination points $D = \{d_1, d_2, \ldots, d_N\}$;
- A set of taxis $T = \{t_1, t_2, \ldots, t_M\}$, $M \leq N$; the maximum number of passengers in the taxi is C_{MAX}, and a function $C{:}T \rightarrow \{0, 1, \ldots, C_{MAX}\}$ indicates how many passengers use the taxi in a trip.
- A symmetric matrix M (dimension $(N+1) \times (N+1)$) with the distances between each one of the geographic points in the problem (one origin and N destinations);
- A cost function c_T given by the constant MF (*minimum fare*) and the cost by distance $c(dist(O, d_i))$.

The problem goal is to find a planning function $f{:}P \rightarrow T$ to transport the N passengers in K taxis ($K \leq M$), determining both the passengers-to-taxis assignment and the order to visit the destinations, minimizing the total cost (TC, Eq. 1).

$$TC = \sum_{t_i}(MF + \sum_{j=1}^{C(t_i)} c(dist(d_{j-1}, d_j)) \quad (1)$$

The proposed formulation minimizes the total cost. Different options can be applied to compute the cost for each passenger [7], including: i) paying for distance and equally dividing the minimum fare, each passenger j in a shared taxi t_i pays $MF/C(t_i)+dist(d_{j-1}, d_j)$; ii) paying a flat fare, disregarding distances, $(MF+\sum_{j=1}^{C(t_i)} c(dist(d_{j-1}, d_j)))/C(t_i)$, etc.

The taxi sharing problem is NP-hard, as it is a variant of the *car pooling* problem [5]. Thus, when dealing with realistic problem instances, heuristics and metaheuristics [6] are the most useful methods to find high-quality solutions in reasonable execution times.

B. Related work

The *taxi pooling* problem is a variant of the *car pooling* problem, which has been studied from different perspectives in the related literature. From the point of view of the taxi companies, Xin *et al*. [8] studied the optimization of the total cost of k taxis serving clients on-line, applying a heuristic that sends the two nearest taxis to attend a request, promoting competition and avoiding underutilization. The results showed that the problem, as a generalization of the k-servers problem [9], holds a competitive ratio of k against an optimal algorithm that knows the entire sequence of requests beforehand.

Balancing the interest of both taxi owners and users, Ma *et al*. [10] proposed a dynamic system for taxi sharing, by combining a search and a planning method to find taxis/assign passengers. A lazy strategy is applied to improve execution times, using previously computed results to delay the shortest path calculation as much as possible. Using a database of 33.000 GPS taxi trajectories from Beijing, the dynamic system reduced the travelled distances up to 13%, while serving 25% more requests in a simulated scenario with 6 requests per taxi.

Closer to the problem we tackle in this article, which focuses mostly on the interests of the customers, Tao *et al*. [11] proposed two heuristic algorithms to optimize taxi ride-sharing costs based on greedy strategies. One algorithm is applied to the one-origin-to-many-destinations problem, and the other one is designed for the many-to-one scenario. The results of a field test of taxi-pooling at Taipei with 10 taxis and 798 passengers show an average matching success rate of 60.3%. However, the fuel savings results are shown only in absolute terms, making it difficult to extract meaningful information to compare with other techniques. The one-to-many problem is the one tackled in this paper, so it is of particular interest to see the performance of the greedy method against the parallel method evolutionary algorithm we propose here.

Besides the academic proposals, there are several on-line applications to solve different version of the car pooling problem, including Carpooling (*http://carpooling.com*), which allows finding trip partners to share costs when people use their own cars. Carpling (*http://carpling.com*) focuses on taxi pooling from the point of view of the companies, finding users with nearby destinations. The solutions computed by Carpling are restricted to trips totally contained in other trips, without considering different trip combinations. There are no proposals of applications solving the taxi sharing problem by explicitly computing routes as proposed in this article.

The analysis of related works indicates that the taxi sharing problem is interesting for the scientific community, having a significant impact on environment protection and economy. There are few user-oriented solutions in literature, so there is room to contribute in this line of research, by proposing efficient methods for planning and reducing vehicular traffic.

In our previous work [12], we proposed a simple EA for taxi sharing optimization that showed a good capability of solving realistic medium-size problem instances, but requiring large execution times. The parallel micro EA we propose in this article is conceived to improve both the quality of results and the computational efficiency of the search.

III. A PARALLEL MICRO EA FOR TAXI SHARING

This section introduces EAs and describes the proposed parallel micro-EA for taxi sharing.

A. Evolutionary algorithms

EAs are non-deterministic methods that emulate the evolution of species in nature to solve optimization, search, and learning problems [13]. In the last twenty-five years, EAs have been successfully applied for solving optimization problems underlying many real applications of high complexity.

Parallel models are a popular option to improve the efficiency and the efficacy of EAs. By splitting the population into several computing elements, parallel evolutionary algorithms (PEAs) allow reaching high quality results in a reasonable execution time even for hard-to-solve optimization problems. The parallel EA proposed in this work is categorized within the *distributed subpopulations* model [14]: the population is split in several subpopulations (*demes*). Each deme runs a serial EA, and the individuals are able to interact only with other individuals in the deme. An additional *migration* operator is defined: occasionally some individuals are exchanged among demes, introducing a new source of diversity in the EA.

Micro-EAs provide an efficient alternative to reduce the execution times when solving on-line optimization problems, by using small populations and restarting procedures. In our research group, micro-EAs have been applied successfully to solve combinatorial optimization problems [15] [16]. The main features of the proposed parallel micro-EA to solve the taxi sharing problem are presented next.

B. Implementation details: encoding and fitness function

Solutions are represented as tuples containing integers between 1 and N, representing the passengers, and $N-1$ zeros to separate passengers assigned to different taxis The order for visiting the destinations is the one specified in the sequence. Fig. 1 shows an example of the solution encoding for an instance with $N = 5$.

The solution encoding has some restrictions/features: i) the number of consecutive (non-zero) integers is limited to C_{MAX}; ii) each integer must appear only once in the encoding; iii) consecutive zeros mean the same than a single one.

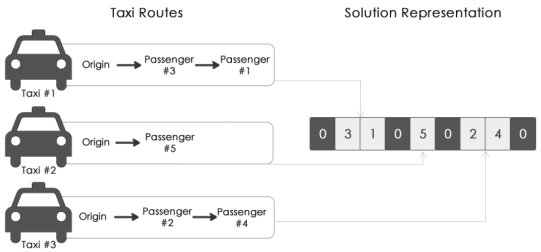


Fig. 1. Example of solution representation for the taxi sharing problem.

The fitness function accounts for the problem objective (cost of the trips). In order to transform the cost minimization problem to a maximization one, we define the fitness function as the inverse of the total cost (defined in Eq. 1).

C. Evolutionary operators

The proposed encoding has specific features and restrictions, so we apply ad-hoc search operators.

Population initialization: the population is initialized using two alternative methods. The *random initialization* uses a constructive algorithm that randomly places the numbers from 1 to N into a tuple initially created with $2N$-1 zeroes. The *greedy initialization* (see next section) applies a random number of perturbations, *i.e.* swaps two elements, over the solution found by a greedy algorithm to solve the problem. Both initialization methods are studied in the experimental evaluation.

Feasibility check and correction process: both initialization methods might violate some of the constraints defined for the solution encoding. Thus, a corrective function is applied to guarantee solution feasibility. The method searches for sequences of non-zero digits larger than the maximum number of passengers allowed per taxi. Then, the correction algorithm locates the first consecutive couple of zeroes, and moves the first zero to a random place at the non-zero sequence, in order to break the invalid group. The search for invalid sequences continues until the end of the solution; at that point, the sequence fulfills all the constraints.

Selection: a *tournament selection* is applied to provide an appropriate selection pressure for the micro populations. Initial experiments confirmed that the standard proportional selection technique does not provide enough diversity to avoid premature dominance and convergence in solutions far from the optimum.

Recombination: we apply an ad-hoc variant of the *Position Based Crossover* (PBX) operator, explained in Algorithm 1. In order to avoid violating the constraints imposed by the solution encoding, the same corrective function used after the initialization is applied on the offspring produced using the PBX crossover. Fig. 2 shows an example of the PBX crossover for two individuals with 5 passengers and $C_{MAX} = 4$.

Mutation: we apply the *Exchange Mutation* operator, which randomly selects and exchanges two positions in the solution encoding. Afterward, the same corrective function mentioned above is applied to fix invalid solutions, if necessary.

Migration: the migration operator considers the demes connected in an unidirectional ring topology, where immigrants replace the worst individuals in the destination deme.

Algorithm 1 Ad-hoc PBX for the taxi sharing problem

1: Randomly select several positions in parent 1.
2: Partially generate the offspring, copying the selected values from parent 1.
3: Mark in parent 2 the positions already selected in parent 1.
4: Select the next non-marked value in parent 2, sequentially from the beginning, and copy it in the first free position in offspring.

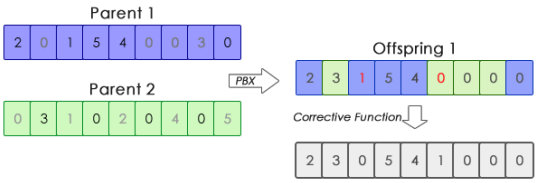


Fig. 2. PBX applied to the taxi sharing problem.

IV. Experimental Analysis

This section reports the experimental analysis of the proposed parallel micro EA to solve a set of realistic instances of the taxi sharing problem.

A. Development and execution platform

The proposed EA was implemented in the C++ programming language, using Malva [17]. The experimental evaluation was performed on a Dell Power Edge server, Quad-core Xeon E5430 processor at 2.66GHz, 8 GB RAM and Gigabit Ethernet, from the Cluster FING high performance computing facility (Universidad de la República, Uruguay, website http://www.fing.edu.uy/cluster) [18].

B. Problem instances

In order to evaluate the proposed EA, a group of real instances were generated. A specific methodological approach for the generation of realistic instances of the problem was used, taking into account the problem restrictions and using services available to gather information about taxi demands, maps, and fares, including:

- *Taxi Query Generator (TQG)*, a tool that uses information from a database of taxi trajectories obtained from GPS devices installed for a week in 10,357 taxis in the city of Beijing, to generate realistic taxi demands. The data are a subset of those used by Ma et al. [10]. TQG produces a list of origin/destination coordinates for individual trips. In order to design useful instances for the problem addressed in this article, we developed a script that groups those trips that originate in nearby locations, thus outputs are according to the one origin-to-many destinations problem formulation.
- *QGIS Desktop*, a free, open source geographic information system, used to create, edit, view and publish geospatial material. QGIS was used to transform the list of coordinates obtained at the output of a script grouping, into a KML (Keyhole Markup Language) file, which allows it to be displayed using Google Maps/Google Earth, to have a visual representation of the origin and destinations instance on a map.

- The *TaxiFareFinder* interface [19], which was used to obtain the cost matrix for each generated instance of the problem, along with the prices corresponding to the minimum fare. Each pair of coordinates for a given instance of the problem is sent to the interface TaxiFareFinder to get the cost between each point.

Following the proposed approach, we created a benchmark set of **24** realistic instances of the taxi sharing problem, with different dimensions: *small* (10–25 taxi requests), medium (25–40 requests), large (40–55 requests), and very large (55–70 requests), to use in the experimental evaluation.

C. Parameter tuning

EAs are stochastic search methods, thus a parameter setting analysis is mandatory. For this purpose, a set of 5 medium-size problem instances (different to the evaluation ones, to avoid biased results) was generated using the aforementioned method. After an initial evaluation, the micro-population size was set to 15. The parameter tuning focused on two key aspects of the EA: crossover (p_C) and mutation probability (p_M). Three candidate values were picked for each one: $p_C \in \{0.6, 0.75, 0.95\}$ and $p_M \in \{0.001, 0.01, 0.1\}$.

For all 9 combinations of candidate values, 20 independent executions were performed for each problem instance, with 100.000 generations in each run. Representative results are shown in Figure 3, reporting the average cost obtained for each combination of candidate values. Because this is a minimization problem, those settings that achieve lower cost values are preferred. The parameter setting results suggest that using $p_C = 0.75$ and $p_M = 0.1$ allows computing the best results for the problem instances solved.

D. Experimental evaluation

The experimental analysis focused on both the quality of the solutions and the performance of the proposed parallel EA.

1) Comparison of initialization strategies: the two strategies used to initialize the population were comparatively studied on the whole set of problem instances, by analyzing the cost values obtained after 100.000 generations using 24 cores to decide which initialization performs the best. We applied the Kolmogorov-Smirnov statistical test with a significance level of 0.05 and we found that the results distributions do not follow a normal distribution. Afterward, the Kruskal-Wallis test was applied to analyze the results distributions for each initialization method.

Table I reports the best, average, and standard deviation cost values obtained when using the greedy and the random initialization in 20 independent executions performed for each problem instance. Overall, both initialization methods computed the same average cost for two small problem instances, the greedy method found the best average in 11 instances, and the random one in 9 instances. Instances where the greedy initialization performed the best, with 95% statistical confidence (p_{value} <0.05 on the Kruskal-Wallis test) are marked in bold.

From the results in Table I, we can assess with statistical confidence that the greedy initialization allows computing better solutions than the random method, especially for large problem instances. Thus, this method was selected for the rest of the experimental evaluation.

TABLE I. INITIALIZATION STRATEGIES: COMPARATIVE RESULTS

instance		*greedy initialization*		*random initialization*	
		best	avg.±std	best	avg.±std
small (10–25)	#1	125.5	125.5 ± 0.0	125.5	125.5 ± 0.0
	#2	168.8	168.8 ± 0.0	168.8	168.8 ± 0.0
	#3	191.0	191.2 ± 0.5	191.0	191.0 ± 0.0
	#4	215.5	215.6 ± 0.1	215.5	215.5 ± 0.0
	#5	297.5	298.4 ± 0.3	297.5	300.9 ± 3.5
	#6	246.1	252.2 ± 2.4	244.1	245.9 ± 2.1
medium (25–40)	#1	336.5	344.5 ± 5.6	336.5	340.5 ± 3.9
	#2	**320.2**	**323.1 ± 3.4**	319.1	324.3 ± 3.6
	#3	**795.5**	**801.2 ± 4.2**	796.1	803.7 ± 4.8
	#4	**351.2**	**357.4 ± 3.4**	351.8	358.6 ± 3.0
	#5	436.8	443.7 ± 3.8	435.6	443.3 ± 5.7
	#6	**359.3**	**367.0 ± 5.0**	360.9	375.2 ± 6.1
large (40–55)	#1	415.0	429.9 ± 7.1	412.6	420.0 ± 3.7
	#2	**304.9**	**319.8 ± 7.5**	306.4	322.0 ± 6.5
	#3	**416.2**	**425.0 ± 4.3**	417.2	431.2 ± 7.1
	#4	362.1	367.7 ± 3.2	361.6	367.0 ± 3.3
	#5	440.3	446.3 ± 2.6	437.6	445.2 ± 4.8
	#6	**553.5**	**562.1 ± 4.3**	554.2	566.5 ± 7.6
very large (55–70)	#1	**626.3**	**637.7 ± 3.6**	630.7	643. ± 6.4
	#2	517.2	524.8 ± 4.5	509.0	520.8 ± 7.7
	#3	**490.0**	**498.4 ± 3.5**	487.3	501.6 ± 7.8
	#4	736.8	744.9 ± 6.1	729.0	742.1 ± 8.0
	#5	**548.7**	**560.6 ± 4.7**	**551.9**	**564.3 ± 7.0**
	#6	**1412.6**	**1424.6 ± 6.1**	1418.7	1430.0 ± 6.1

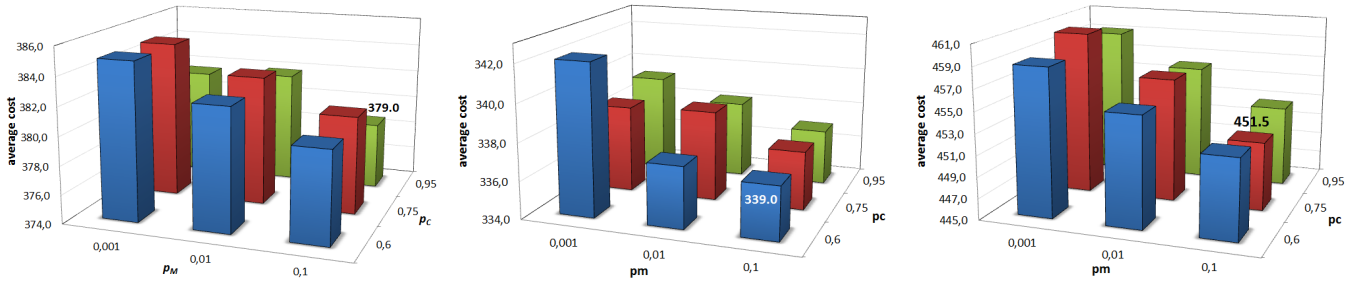
2) Comparison against the sequential EA: Table II reports the average costs obtained for each instance dimension, using the pμEA—with 8, 16, and 24 cores—and the sequential EA. From the results, it is clear that using more generations leads to better solutions. In addition, pμEA has a good scalability behavior, as cost results improve when using additional cores, up to 1.6% when comparing pμEA–24 cores vs. pμEA–8 cores on a large problem instance.

The results in Table II demonstrate that pμEA outperforms the sequential EA on every instance in the test set, with significant cost improvements (according to the Kruskal-Wallis test). Over all instances, the average cost improvements computed by pμEA over the sequential EA were 4.3%. In the best case, instance #1 in the set of large problem instances, an improvement of up to 8.5% was achieved against the average cost obtained by the sequential EA. The full experimental results are available at www.fing.edu.uy/inco/grupos/cecal/hpc/AG-Taxi.

TABLE II. COMPARISON OF PμEA USING DIFFERENT NUMBERS OF SUBPOPULATIONS AND THE SEQUENTIAL EA

instances	*algorithm*	*generations*				
		10000	25000	50000	75000	100000
small (10–25)	pμEA–8 cores	212.2	211.2	210.1	209.7	209.5
	pμEA–16 cores	210.6	209.7	209.2	208.9	208.8
	pμEA–24 cores	209.9	209.3	208.9	208.7	208.6
	sequential EA	219.3	217.6	216.5	215.6	215.2
medium (25-40)	pμEA–8 cores	454.9	449.3	447.5	446.9	446.5
	pμEA–16 cores	449.5	444.0	442.2	441.6	440.7
	pμEA–24 cores	448.0	442.7	441.0	440.0	439.5
	sequential EA	475.6	468.7	466.3	465.2	464.6
large (40–55)	pμEA–8 cores	454.2	438.8	433.6	432.2	431.4
	pμEA–16 cores	450.9	435.4	428.9	427.6	426.9
	pμEA–24 cores	448.7	432.4	427.6	425.8	425.1
	sequential EA	475.7	456.6	449.3	447.2	446.6
very large (55-70)	pμEA–8 cores	788.2	759.3	745.8	741.4	739.5
	pμEA–16 cores	781.5	752.8	741.0	736.7	734.9
	pμEA–24 cores	777.9	750.9	737.9	733.6	731.8
	sequential EA	815.0	782.2	766.4	762.7	761.1

3) Comparison against an intuitive greedy algorithm: A greedy strategy was implemented in order to compare the performance of pμEA against a traditional intuitive heuristic.



(a) Instance #1 (b) Instance #2 (c) Instance #3

Fig. 3. Average cost obtained with different parameter configurations on three different problem instances

The greedy method (Algorithm 2) sequentially adds the passenger whose destination is closer to the origin in a taxi, until the taxi is full. In the case where the cost of adding one passenger to the current taxi is greater than the one obtained by assigning a new taxi to serve that passenger request, the current taxi is full, and a new one is formed. The algorithm ends when every passenger is assigned to a taxi. This method emulates an intuitive approach to solve the problem by taking local optimum decisions [11]. A similar strategy can be expected from a group of human users trying to solve the problem.

Algorithm 2 Intuitive greedy algorithm for taxi sharing

```
taxi = 1; counter = 0; {Init}
while passengers to assign do
  if (counter == 0) then
    p1 = nearestDestinationfromOrigin();
    addPassenger(p1,taxi,solution);
    counter++;
  else if (counter ≥ C_MAX ) then
    counter = 0; {Current taxi is full}
    taxi++;
  else
    p2 = nearestLocation(p1);
    if (cost(p1,p2) ≤ cost(origin,p2) + MF) then
      addPassenger(p2,taxi,solution);
      p1 = p2;
      counter++;
    else
      taxi++; {Start a new taxi}
      counter = 0;
    end if
  end if
end while
return solution;
```

Since the greedy algorithm is very fast, it is important to evaluate how long it takes the pμEA to outperform it. Figure 4 reports the average time pμEA need to outperform the greedy solution by 5%, 10%, 15% and 20% (in the best case for each instance). Additionally, the average overall improvement and average time to reach 100.000 generations is displayed for each problem instance (upper row, in red).

The greedy algorithm executes in a negligible amount of time, but the results in Figure 4 show that pμEA is able to improve the greedy by 5% almost instantaneously, and improvements of about 10% are computed in most small/medium/large cases in a few seconds. pμEA also computes solutions with larger improvements over greedy (up to 36.1%), in execution times that strongly depends on the instance and dimension.

In the best case for each problem dimension, pμEA improved the results computed using the greedy algorithm by **36.1%** in small instance #2, **27.4%** in medium instance #5, **27.2%** in large instance #4, and **25.0%** in very large instance #3. The full results of the comparison are available at www.fing.edu.uy/inco/grupos/cecal/hpc/AG-Taxi

The previous results demonstrate that the proposed pμEA is an accurate and very efficient tool for cost optimization in the taxi sharing problem. The reduced execution times to compute significant improvements over traditional techniques make pμEA an appropriate method for on-line optimization, to be used in application-oriented websites such as the one we are currently deploying in http://www.mepaseaste.uy.

V. Conclusions and Future Work

This work has presented the design and implementation of a parallel micro evolutionary algorithm for the one-origin-to-many-destinations taxi pooling problem.

Vehicle sharing is an interesting topic nowadays, mainly because its impact in economic cost and environmental protection, and the scientific community has been studying diverse variants of the car pooling problem in the last years. In this work, we study a specific variant of the taxi sharing problem and propose a parallel micro EA to solve it.

The proposed algorithm was conceived to provide accurate and efficient solutions for the problem, in order to improve the previous results computed by the sequential EA presented in [12]. By using a distributed model and micro populations, the proposed pμEA provides an improved search pattern that allows solving the planning problem in online mode.

Indeed, in the experimental analysis performed over a benchmark set of 24 realistic problem instances generated using real GPS data from taxis in the city of Beijing, pμEA was able to compute significant improvements over both the sequential EA and an intuitive greedy algorithm to solve the problem. Regarding the cost of solutions, improvements up to 8.5% (4.3% in average) were achieved over the sequential EA and up to 36.1% (18.2% in average) over the greedy method. Furthermore, significant improvements are computed in very fast execution times, making the proposed optimization technique an appropriate tool for online taxi sharing planning

The main lines for future work are focused on including other user-oriented information in the problem formulation, such as online traffic data and taxi availability, in order

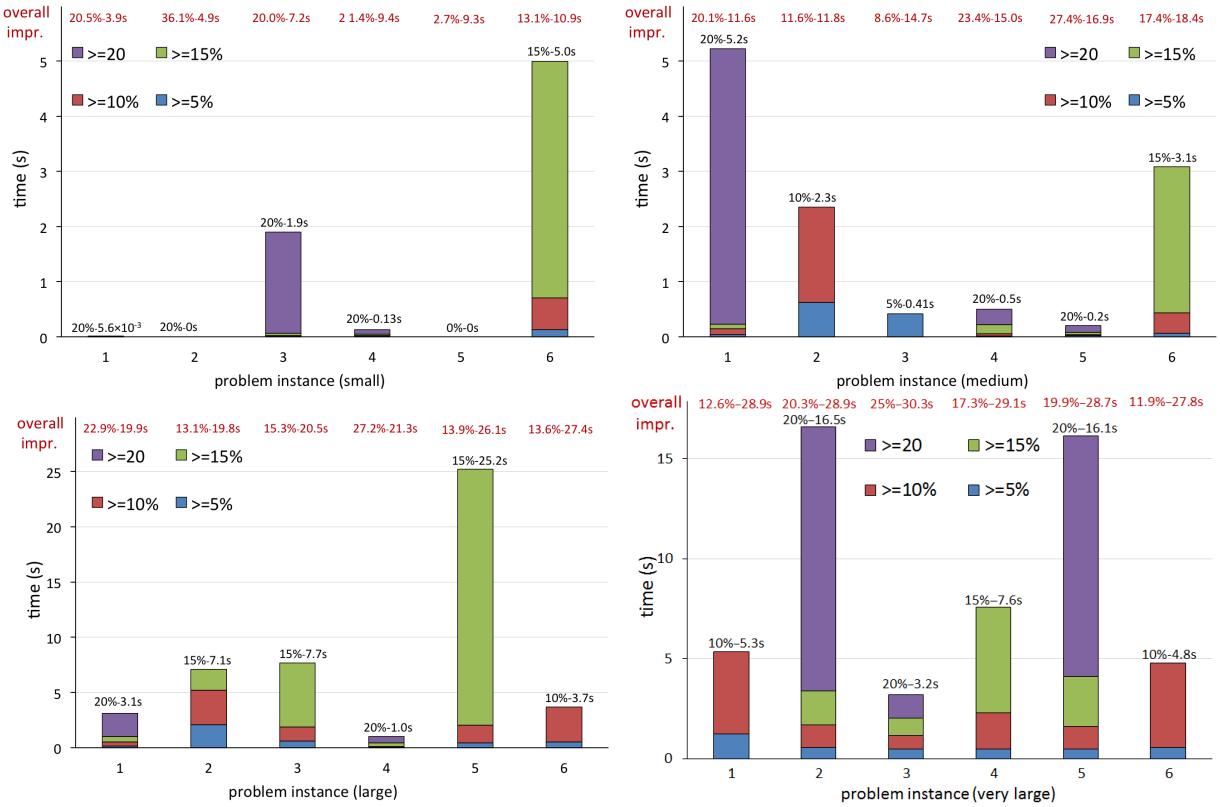


Fig. 4. pμEA comparison against the greedy algorithm: best and average cost improvements and average execution times.

to provide a more realistic planning. The proposed pμEA will be incorporated in our website for taxi planning. Other optimization techniques, including multiobjective EAs should also be evaluated to solve the problem.

ACKNOWLEDGEMENT

The research reported in this article was partly supported by ANII and PEDECIBA, Uruguay.

REFERENCES

[1] N. Fellows and D. Pitfield, "An economic and operational evaluation of urban car-sharing," *Transportation Research Part D: Transport and Environment*, vol. 5, no. 1, pp. 1–10, 2000.

[2] E. Ferrari, R. Manzini, A. Pareschi, A. Persona, and A. Regattieri, "The car pooling problem: Heuristic algorithms based on savings functions," *Journal of Advanced Transportation*, vol. 37, pp. 243–272, 2003.

[3] R. Katzev, "Car sharing: A new approach to urban transportation problems," *Analyses of Social Issues and Public Policy*, vol. 3, no. 1, pp. 65–86, 2003.

[4] "Ride-Sharing Services Grow Popular in Europe," The New York Times, By Eric Pfanner, Published: Oct. 1, 2012, [Online 02-2014] http://www.nytimes.com/2012/10/01/technology/ride-sharing-services-grow-popular-in-europe.html.

[5] A. Letchford, R. Eglese, and J. Lysgaard, "Multistars, partial multistars and the capacitated vehicle routing problem," *Mathematical Programming*, vol. 94, no. 1, pp. 21–40, 2002.

[6] S. Nesmachnow, "Metaheuristics as soft computing techniques for efficient optimization," in *Encyclopedia of Information Science and Technology*, M. Khosrow-Pour, Ed. IGI Global, 2014, pp. 1–10.

[7] "How to Split a Shared Cab Ride? Very Carefully, Say Economists," The Wall Street Journal, Dec. 8, 2005, [Online 06-2014] http://online.wsj.com/news/articles/SB113279169439805647.

[8] C.-L. Xin and W.-M. Ma, "Scheduling for on-line taxi problem on a real line and competitive algorithms," in *Proc. of the Int. Conf. on Machine Learning and Cybernetics*, 2004, pp. 3078–3083.

[9] E. Koutsoupias, "The k-server problem," *Computer Science Review*, vol. 3, no. 2, pp. 105–118, 2009.

[10] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *IEEE 29th Int. Conf. on Data Engineering*, 2013, pp. 410–421.

[11] C.-C. Tao and C.-Y. Chen, "Heuristic algorithms for the dynamic taxipooling problem based on intelligent transportation system technologies," in *4th Int. Conf. on Fuzzy Systems and Knowledge Discovery*, 2007, pp. 590–595.

[12] G. Fagúndez, R. Massobrio, and S. Nesmachnow, "Resolución en línea del problema de viajes compartidos en taxis usando algoritmos evolutivos," in *Conferencia Latinoamericana de Informática*, 2014.

[13] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of evolutionary computation*. Oxford University Press, 1997.

[14] E. Alba, G. Luque, and S. Nesmachnow, "Parallel metaheuristics: Recent advances and new trends," *Int. Transactions in Operational Research*, vol. 20, no. 1, pp. 1–48, 2013.

[15] S. Nesmachnow, H. Cancela, and E. Alba, "A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling," *Applied Soft Computing*, vol. 12, no. 2, pp. 626–639, 2012.

[16] S. Nesmachnow and S. Iturriaga, "Multiobjective grid scheduling using a domain decomposition based parallel micro evolutionary algorithm," *Int. Journal of Grid and Utility Computing*, vol. 4, pp. 70–84, 2013.

[17] "The Malva Project: A framework for comutational intelligence in C++," [Online 05-2014] https://github.com/themalvaproject.

[18] S. Nesmachnow, "Computación científica de alto desempeño en la Facultad de Ingeniería, Universidad de la República," *Revista de la Asociación de Ingenieros del Uruguay*, vol. 61, pp. 12–15, 2010.

[19] "TaxiFareFinder API," [Online 05-2014] http://www.taxifarefinder.com.