Received May 15, 2020, accepted May 23, 2020, date of publication May 26, 2020, date of current version June 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2997812

A Parallel Genetic Algorithm Framework for Transportation Planning and Logistics Management

DMITRI I. ARKHIPOV[1], DI WU[1,2], (Member, IEEE), TAO WU[3], AND AMELIA C. REGAN[1], (Member, IEEE)
[1]Department of Computer Science, University of California at Irvine, Irvine, CA 92697-3435, USA
[2]State Key Laboratory of Advanced Design and Manufacturing of Vehicle Body, Hunan University, Changsha 410082, China
[3]Key Laboratory of Geospatial Big Data Mining and Application, Hunan Normal University, Changsha 410081, China

Corresponding authors: Di Wu (e-mail: dwu@hnu.edu.cn) and Amelia C. Regan (e-mail: aregan@ics.uci.edu)

This work was supported in part by the University of California Transportation Center, in part by the National Natural Science Foundation of China under Grant 61972145 and Grant 61932010, in part by the National Key Research and Development Program of China under Grant 2019YFB1405703, in part by the Huxiang Youth Talent Program under Grant 2018RS3040, and in part by the Open Research Fund of State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing under Grant 19I05.

ABSTRACT Small to medium sized transportation and logistics companies are usually constrained by limited computing and IT professional resources on implementing an efficient parallel metaheuristic algorithm for planning or management solutions. In this paper we extend the standard meta-description for genetic algorithms (GA) with a simple non-trivial parallel implementation. Our parallel GA framework is chiefly concerned with the development of a straightforward way for engineers to modify existing genetic algorithm implementations for real transportation and logistics problems to make use of commonly available hardware resources without completely reworking complex, useful and usable codes. The framework presented at its parallel base is a modification of the primitive parallelization concept, but if implemented as described it may be gradually extended to fit the qualities of any underlying problem better (via the adaptation of the merging and communications functions).We present our framework and computational results for a classical transportation related combinatorial optimization problem – the traveling salesman problem with a standard sequential genetic algorithm implementation. Our empirical analysis shows that this simple extension can lead to considerable solution improvements. We also tested our assumptions that the framework is easily implemented by an engineer not initially familiar with genetic algorithms to implement the framework for another minimum multiprocessor scheduling problem. These case studies verify that our framework is better than primitive parallelization because it gives empirically better results under equitable conditions. It also outperforms fine grained parallelization as it is easier and faster to implement.

INDEX TERMS Parallel metaheuristics, genetic algorithm, transportation planning, logistics management.

I. INTRODUCTION

Genetic algorithms (GA) are an iterative search method in which new answers are produced by combining two predecessor answers and mimicking the process of natural selection [1]. Its metaheuristic can routinely generate useful solutions to optimization and search problems, therefore GA has been widely used in transportation planning [2]–[5] and logistics operations management applications and software systems [6]–[8]. Some current relevant applications include pavement lifecycle analysis [9], additive manufacturing [10], accident emergency response [11] and land use planning [12]. Other frameworks for transportation analysis might benefit from the addition of parallel migration implementation in scenarios such as traffic congestion [13]–[15], vehicular sensing [16]–[18], activity planning [19]–[21], mobility management [22]–[24], spatial-temporal modeling [25]–[27] and others [28]–[31].

The associate editor coordinating the review of this manuscript and approving it for publication was Guangdong Tian.

A. MOTIVATIONS

Genetic algorithm is a common tool that has been employed in many transportation and logistics settings. However, in reality, for small to medium sized manufacturing and distribution companies, or transportation planning agencies that have limited computing and IT staff resources, running a genetic algorithm ''black box'' is usually time-consuming and inefficient on generating reasonable results for planning operations or scheduling of processes or logistics operations. Since such organizations are not large enough to employ high level programmers or engineers trained in optimization methods, they might want to improve their planning or scheduling system without spending an inordinate amount of their limited IT budget on consulting or additional personnel. Furthermore, such companies might not want to ''fix what is not broke'' and risk taking on life-cycle costs for a new software system. Though existing GA frameworks can offer some fine-grain parallelization for transportation planning and logistics management, they are indeed painstakingly constructed to achieve optimal performance for a generic GA (as discussed in Section II). In addition, these frameworks often require engineers to heavily modify or even completely rewrite an already existing GA implementation.

To avoid the complexity and extra work for entry level engineer to run GA for reasonable transportation and logistics results, an inexpensive GA framework is needed to parallelize an existing genetic algorithm more intelligently than primitive parallelization, but with less effort than recoding in a parallel GA framework [32]. Our system provides a way for any reasonably well trained computer science or engineering student or IT professional to extend a standard sequential genetic algorithm solver in a simple, yet non-trivial parallel framework that does not require extensive re-working of the system nor extensive understanding and testing of problem parameters.

B. OUR APPROACH

Many good parallel meta-heuristics and specifically parallel genetic algorithms have been explored by other researchers. Our work is chiefly concerned with the development of a straightforward way for engineers to modify existing genetic algorithm implementations for real industrial or scientific problems to make use of commonly available hardware or cloud-based resources [33] without completely reworking complex, useful and useable codes. Given limited computing resources in most small to medium sized transportation or logistics companies, we propose a simple but non-trivial parallel implementation of genetic algorithm for those companies to achieve inexpensive and efficient transportation planning and logistic management solutions.

Specifically, we extend the standard meta-description of a genetic algorithm to a parallel environment by assuming that the algorithm is run on n separate but connected processes on an n-core multiprocessor machine. We introduce a functional framework consisting of two functions. One of these functions encodes a communications function indicating the time intervals at which sequential genetic algorithm solvers running on independent processors will communicate, and which sets of such solvers will intercommunicate with one-another at such time intervals. The second functions encodes a merge/synchronization policy which defines the nature of the communications between solves which communicate. In particular the merge/synchronization policy defines what data is passed between processes what operations on this data must be performed as part of the communication. Collectively these two policies define a large subset of all possible migration policies, within this framework we implements a simple example and evaluate it empirically.

Our functional framework is proposed for describing solution migration in a parallel genetic algorithm system. This GA framework can improve parallel performance through utilizing multi-core multi-processor resources [34], [35] and increasing quality of GA solution per time unit while migrating parallel base. The framework presented at its base is a simple modification of the primitive parallelization concept, but if implemented as described it may be gradually extended to fit the qualities of any underlying problem better (via the adaptation of the merging and communications functions). Because we extend the general meta-description of a genetic algorithm to incorporate solution migration in an easy-to-implement way, our parallel GA framework is simple enough to be implemented by any competent third or fourth year computer science or engineering student, or entry level engineer.

The rest of this paper is organized as follows. Section II gives an overview of related work on parallel GA. Section III describes our parallel GA framework. Section IV studies the traveling salesman problem with our framework. Section V evaluates the minimum multiprocessor scheduling problem using our framework. Section VI concludes our paper.

II. RELATED WORK

The literature on parallel GA's and parallel evolutionary algorithms, and more generally parallel metaheuristics is vast. We mention only a few key review papers here. For extensive helpful discussions please see the work by Cantú-Paz and Goldberg [36], [37]. In the language used in those and related papers, our framework falls into a multi-deme scheme where each of the n processors contains a deme and there are varying degrees of migration and synchronization between these. However unlike the multi-deme scheme described in [36] we present a framework centering around on the definition of two functions (communication and merge) within which the multi-deme scheme of [36] is a particular case.

When parallelizing a genetic algorithm there are two primary approaches. First, a designer who is starting from scratch may choose to design the GA to make use of ''system parallelization'' at a fine grained level and use a predefined parallel genetic algorithm framework such as the grid computing framework by Lim *et al.* [38], Cahon et al's well known ParadisEO (see for example [39]), or Bleuler et al's PISA (see for example [40]). System parallelization approach forces the developer to conform to the constraints and interfaces imposed by the framework, though in the end much greater benefits will be realized, relative to more primitive parallelization. Pursuing such systemic methods achieves a "state of the art parallel genetic algorithm" at the cost of a potentially significant amount of developer time.

Second, one might try "primitive parallelization". In this technique n separate threads or processes are created. Each of the n processes executes m iterations of a sequential genetic algorithm. After these m iterations from each of the genetic algorithm solvers, the union of the solutions is taken and then the best, or possibly the k best solutions are selected. Pursuing primitive parallelization achieves hardware exploitation at the cost of an insignificant amount of developer time. However, solutions derived through primitive parallelization are typically inferior to those achieved by systemic parallelization.

The problems associated with the first approach is that if fine grained parallelism is used, the structure of the original solution may not scale gracefully with changing representations of the problem. Also, fine grained parallelism is difficult to discover, implement, and test. If a parallel GA framework is used then one is bound by the decisions made by the framework developers, the model of computation they use, and the interfaces they specify. In addition, if one already has a sequential GA developed and wishes to parallelize it quickly, both fine grained parallelism and implementation of a pre-defined GA framework may be too costly in terms of developer time. Therefore, given limited computing resources and IT professionals in most small to medium sized transportation or logistics companies, a tailored implementation of GA with primitive parallelization is more efficient for a reasonable solution.

III. PARALLEL GA FRAMEWORK

In this work we propose an extension of the meta description of genetic algorithms, that would allow for a convenient process for converting an existing sequential GA, and the problem it solves, into a parallel GA.

A. GENETIC ALGORITHMS

Genetic (and evolutionary) algorithms refer to a broad class of heuristic algorithms that have certain characteristics in common, which perform better in convergence and adaptability analyses than algorithms such as article swarm, random forests, grey wolf algorithm, particle filter. Most of those characteristics relate to the idea that these algorithms are intended to model the "survival of the fittest" mechanism of biological evolution. Genetic algorithms demand that solutions to the problem may be represented in some binary format, that two or more such solutions can meaningfully be merged, and that an objective function value can be evaluated for each such solution.

The simplest meta-description for a sequential genetic algorithm has the following steps:

1) Encoding
2) Initial Population Generation
3) Repeat until termination criteria are met
    a) Evaluation
    b) Crossover
    c) Mutation
4) Decoding

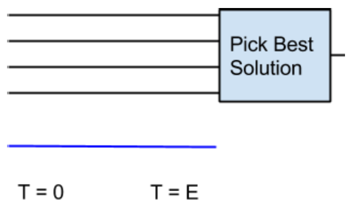

FIGURE 1. Primitive parallel base implementation.

More detailed description of genetic and evolutionary algorithmic approaches, and the applications of such approaches can be found in Mitchell [41].

B. PRIMITIVE PARALLELIZATION OF PARALLEL BASE

The problems associated with primitive parallelization stem from the fact that separating the parallel processes results in less effective overall solutions. A high level description of this approach is presented in Figure 1, where T represents a time line, E symbolizes the terminating time of the process, and each black line represents the parallel execution of a genetic algorithm thread. If the populations can undergo some limited level of mixing without significantly increasing the complexity of the solution framework, the quality of the solutions will typically increase. Exploring ways to leverage this mixing is a primary purpose of this research.

Even at the seemingly trivial level described above, haphazard primitive parallelization may not produce the quality of solution expected or desired. For example, dividing a set number M of GA generations across several independently and concurrently running processes (say P such processes), then choosing the best of these solutions will more often than not result in a worse quality solution than running all M generations on a single process (this of course assuming no shared memory is used). One might expect that the quality of solution generated in either way might be similar but with a speedup factor of P, the reason why this is not so can be seen from the experiment of setting $M = P$. In that case the course of the algorithm is simply picking one of $M = P$ initially generated solutions and no genetic algorithm is run at all. Cases where $P < M$, exhibit a similar loss of performance though of course less extreme. The naïve expectation is not justified because it assumes that each run of the algorithm will on average perform a similar amount of work as any other run, this is not the case.

It is clear that iterations of a GA which start with a more robust population will produce a yet more robust population on average. It is also clear that a population generated by some k runs of a GA will be on average more robust than a population generated by n runs, where $n < k$. The increase in robustness of each successive generation is thus dependent on the fitness of each previous generation, thus a simple conversion of temporal iterations for parallel iterations is not equivalent. In other terms making such a trade-off amounts to moving towards diversification from intensification. This effect was very prominent in our experiments quickly leading us away from this naïve expectation.

C. MIGRATING PARALLEL BASE

We extend the meta description of a genetic algorithm given earlier to a parallel environment by assuming that the algorithm is run on n separate but connected processes on an n-core multiprocessor machine and that these processes communicate with each other in the following ways: We supply each processor with a Problem Instance Core or PIC. This contains the problem instance of interest, an initial population of solutions, all of the relevant genetic algorithm parameters (population size, survival rate, elitism rate, mutation rate, etc.) as well as any additional parameters such as switching mechanisms between crossover or mutation methods. Additional parameters are sometimes encorporated by GA designers when they wish to incorporate aditional complexity into the crossover and mutation phases of their GA.

The populations initially generated for each PIC must of course be distinct; this is the origin of the higher diversity in our framework. In other words, while the other parameters of each generated PIC (population size, mutation rate, etc.) may be identical in each PIC generated from a problem instance, the actual members of the population must not be identical to those generated in another PIC. Depending on the details of the implementation of the core genetic algorithm, additional parameters may be passed. The PIC must contain all data necessary for the base GA implementation to execute.

The initial parameter settings at each PIC are again dependent on the details of the base GA implementation and operator experience. If for example the operator has identified that a given parameter set works best for instances of their problem they may choose to give such a setting to each of the GA processes in the system set up in this framework. If on the other hand the operator is experimenting with several parameter sets they may well choose to create several PICs (one for each parameter set) and assign each PIC to a GA solver operating within the described system. The construction of the system thus allows the operator to test alternative parameter sets and harness the benefits of each within a single run of the full system.

The processors then run the GA solver on these inputs in the first time step, and then engage in a migration/synchronization phase, which is followed by iteratively repeating step 3 (see Section III-A) in further time steps until the overall stopping criteria are met. We show this diagrammatically in Figures 2. Note that the precise specification of the communications policy is fairly arbitrary. We have selected a policy here that is simple to explain and implement, namely, at the end of each GA run, results obtained on each processor k are passed to processor $k+1$ except for the n'th processor the results from which are passed to processor 1. In the most general form the only constraint for a communications policy used in the system is that each processor receives

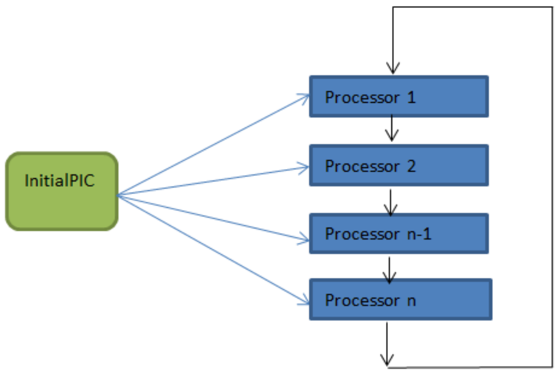

FIGURE 2. Information flow at time step 0 (left side) and flow of migration and synchronization (right side).

a PIC at the beginning of each time step, generally that PIC will be a PIC resulting from the merging of 1 or more PICs present in the system at the previous time step. These results are then merged with the results on the receiving processor. This merge/synchronization step is also quite general. The K best individuals in the population are chosen to remain in the merged PIC, where K is population size determined for the PIC resulting from the merge operation.

We present this as a simple and implementable solution with the understanding that experienced engineers might prefer to come up with their own variation on this scheme. The most general form of a merge/synchronization policy within this system is simply a function which maps more than one PICs to a single PIC. This allows the operator to implement any parameter tuning strategy that they desire, generating parameters for the resulting PIC from the parameters within the input PICs. Inputs needed for parameter tuning would be generated and maintained within each core GA and be used by the function implementing the merge/synchronization policy at each processor, at each merge/synchronization phase. Another component of this process that must be considered is the periodicity or number of synchronization/migration phases that occur per full problem execution. This is another aspect that it is left for the operator to design, in the simplest case and the policy chosen in this work the operator will chose a fixed number of generations after which a synchronization/merging phase will occur. The synchronization/merging phase may be triggered by a dynamically satisfied condition, or as in this work may be assigned statically before the algorithm is executed. In evaluating our approach we examine the quality of solution generated for a given amount of time, while much of the work on parallel algorithms concentrates on speed-up of the parallel approach relative to the sequential approach we look at the difference in end solution quality for a fixed set if iterations.

D. SOLVING LOCAL MINIMA

One common problem with heuristic approaches in general, and GAs in particular is that they get stuck in local minima and then are not able to break out of these. There are many approaches used to address this. One involves selection strategies that reduce the likelihood of super fit individuals being constantly selected during the selection phase. Another is to periodically add diversity by injecting randomly generated solutions into a working population. In the case of a GA the problem cannot be addressed by periodically injecting randomly generated solutions into a working population. The reason for this is that the randomly generated solutions are too uncompetitive with the already partially evolved solutions in the populations and they will quickly be out-competed and killed off.

Typically a GA will ultimately get stuck at a local minimum because some solution or group of solutions become so dominant that no child solutions are given a reasonable chance to produce offspring, and if they do, these offspring will be similar to these dominant solutions. Techniques such as choosing appropriate selection strategies serve to increase the number of generations before such premature convergence. Keeping each of the populations sequestered to separate threads or processes, as we would in primitive parallelization, can serve to preserve solution diversity. However since the threads/processes are executing simultaneously with a different randomly generated populations they are likely to generate different solutions. The migration technique serves to disrupt steady state populations by introducing solutions from other steady state demes. In this paper we chose somewhat arbitrary periods for introducing this disruption, however a more rigorous approach is possible. Vishnoi [42] gives a bound on population mixing-times under certain assumptions, genetic algorithm designers may use this bound to determine periodicity of migrations in our framework.

This opens up an opportunity. If migrants cross from thread to thread at certain time intervals then it is likely that they will not be out-competed by the population already present at the arriving thread. That is unless of course the migrants start arriving too often, meaning that the genetic pool at one process/thread is more or less identical to the pool at another, and solutions generated by mixing these pools will not diverge from the populations already in them. A parameter setting or a dynamically determined variable may determine the frequency of such migration (in our research a fixed period was defined for each experiment). Of course since the threads are run in parallel, each migration will necessitate a synchronization of the threads and thus introduce a small delay in overall execution. Results indicate that this delay is warranted for a small number of synchronizations.

E. FRAMEWORK COMPONENTS AND WORKFLOW

The common availability of multicore machines, clustered computers and cloud computing, and the advantage of keeping distinct groups of individuals with similar levels of evolution as breeding stock leads to a very general and simple to implement framework for GAs. The components of such a framework are 1) the initial population and initial parameter generation mechanisms. 2) The specifics of the sequential GA implementations used in the framework. 3) A setting to statically or dynamically determine the synchronization period 4) the communications function indicating which two PICs will be merged, and on which GA process the resulting PIC will be run after synchronization. 5) The PIC merge function which will map n PICs to one PIC. 6) And, finally, a termination condition for the framework as a whole. Clearly a sequential GA can be easily implemented in this framework. The ease and generality of the framework allow many diverse parallel GAs to be developed without significant code alteration to the core sequential GAs.

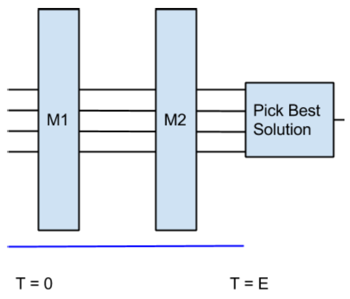

FIGURE 3. **Merging phases.**

Algorithm 1 Parallel GA Framework
```
1: procedure MigratingGA
2:     $P \leftarrow$ Problem instance for original sequential GA
3:     $GA \leftarrow$ The original sequential GA
4:     $POP[0 \ldots N-1] \leftarrow GenerateNinitialpopulations$
5:     parfor $k \leftarrow 1, N$ do
6:         $SOL[k] \leftarrow GA(POP[k])$
7:     end parfor
8:     for $(i, j) \in DOM(f)$ do
9:         $POP[f(i, j)] \leftarrow Merge(SOL[i], SOL[j])$
```

Algorithm 1 describes the workflow of our parallel GA framework, where $DOM(f)$ indicates the domain of the communication function f, *i.e.* the pairs of GAs that communicate as per the communications policy. $f(i, j)$ is dependent on the communications function chosen, $Merge(\ldots)$ represents the merge function mentioned earlier.

At a high level the $Merge(\ldots)$ approach is described in Figure 3, where T represents a time line, E symbolizes the terminating time of the process, each black line represents the parallel execution of a genetic algorithm thread, and $M1$ and $M2$ are both merge phases. As described previously in merge phases solutions gradually migrate from one population to another according to the communications and merging functions.

IV. CASE STUDY I: TRAVELING SALESMAN PROBLEM

We began our evaluation by testing our framework on one of the most often used combinatorial optimization problems in transportation planning and logistics management, the Traveling Salesman Problem (TSP) [43]. General GA and its parallel algorithms have been widely used to address many TSP variants [44]. Like our parameter choices and migration/synchronization scheme, we merely use TSP as a simple exploratory example (a proof of concept). Of interest here is not so much the solutions found under the different scenario setups, but the relative performance under these setups. Clearly if the goal were to improve the absolute solution quality than any number of intermediate or final route improvement heuristics could and would be added. However, to fairly place our particular implementation's absolute performence relavtive to published results on tested problem instances we run a simple 2-OPT heuristic on finally generated solutions (see Table 2).

A. TEST ENVIRONMENT

The framework and experiments were written the Python. The tests were performed on a 24 GB memory machine with a 2.4 GHz processor with the following caches (a 32 Kb 8-way set associative level one cache; 256Kb 8-way set associative level 2 cache; and a 6144Kb 12-way set associative level 3 cache). Four threads were used in all experiments.

B. PROBLEM INSTANCES

We performed extensive testing on TSP instances. These problem Instances were drawn from the well-known TSPLIB problems [45]. We present the results for problems instances eil51, st70, kroA100, pr226 and pr1002 where the numbers in the problem names represent the number of nodes in each problem instance.

Each test was run twelve times under all GA system configurations. In each of these configurations there was either one GA solver working synchronously or four solvers working in parallel. The details of the three final GA system configurations are described below:

1) (SSB) Standard Sequential Base: All iterations are run on a single GA machine. No migration, no synchronization takes place. We arbitrarily selected to run $I = 1024, 2048, 4096$ iterations.
2) (PPB) Primitive Parallel Base: four GA solvers are run in parallel, each runs the same number of generations/iterations as in the base case. At the end of this parallel phase the best solution from the four solution sets is returned.
3) (MPB) Migrating Parallel Base, four GA solvers are run in parallel. They are synchronized the number of times stated in Table 1. Each solver in the system operates over the full number of iterations. Thus at the end of the run in total of $4I$ iterations have taken place in the system, where $I = 1024, 2048, 4096$ as specified in Table 1. However because synchronization time is negligible, both cases take approximately the same amount of time when run on a machine with 4 cores (as was done in this experiment).

The time required for each system to run can be expressed in terms of the time required for SSB, B, with additional terms

TABLE 1. Relative performance of SSB, PPB and MPB.

Problem Name	Heuristic	Percent Over Optimal(%)	Iterations	Synchronizations
eil51	SSB PPB MPB	3.31 2.41 2.19	1024	4
st70	SSB PPB MPB	3.94 2.04 1.64	2048	8
eil76	SSB PPB MPB	4.91 3.7 3.61	2048	4
eil101	SSB PPB MPB	6.53 5.55 4.82	2048	12
kroA100	SSB PPB MPB	6.84 4.19 2.49	2048	12
kroB100	SSB PPB MPB	4.71 4.51 3.29	2048	12
pr226	SSB PPB MPB	6.93 3.75 3.43	4096	12
pr1002	SSB PPB MPB	15.42 15.08 13.21	4096	12

ϵ_1 and ϵ_2; where ϵ_1 and ϵ_2 are negligible in relation to B and $\epsilon_1 < \epsilon_2$.

$$Time(SSB) < Time(PPB) < Time(MPB) \tag{1}$$
$$B < B + \epsilon_1 < B + \epsilon_2 \tag{2}$$

The additional terms ϵ_1 and ϵ_2 represent the time required to select the best solution in PPB and the time for synchronization for MPB. The number of iterations performed in PPB and MPB is identical. In each case we finish each of the twelve separate runs by running an inexpensive 2-opt improvement heuristic. We do this simply so that we can demonstrate performance within the range of heuristics that are tuned specifically to TSP problems – the relative performance of the three heuristics is unchanged. So, in each case we incur some additional time for the $O(n^2)$ 2-opt improvement. This time is not uniform, as it varies across solutions, even for the same problem instance, but it is nearly uniform across these three similar solutions for the same problem instance. Therefore, it B_f is the final time required for SSB heuristic to run, including the 2-opt improvement, then the relative solution times are approximately expressed by:

$$B_f < B_f + \epsilon_1 < B_f + \epsilon_2 \tag{3}$$

Comparing the solution quality of SSB, to PPB and MPB over a number of arbitrarily selected problems from the TSP lib, it is easy to see that MPB outperforms both of the others. Specifically, MPS outperforms PPB. This shows the advantage of using the technique described. Table 1 and Figure 4 show these results (in the figure we leave out pr1002 so that the graph can more easily show the differences).

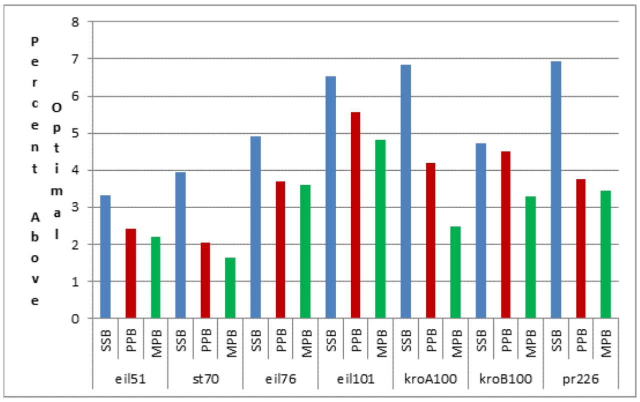

FIGURE 4. Average performance of sequential, primitive parallel and migrating parallel GAs.

TABLE 2. Performance of our GA vs. recently published results.

Problem Name	Source	AVG	Opt	Percent Deviation From Optimal(%)
eil51	This Paper	435.3	426	2.19
eil51	Reference [47]	440.3	426	3.36
eil51	Reference [48]	433.05	426	1.65
st70	This paper	686	675	1.64
st70	Reference [49]	685	675	1.48
eil76	This paper	557.417	538	3.61
eil76	Reference [47]	543.2	538	0.97
eil76	Reference [48]	562.93	538	4.63
eil101	This paper	659.33	538	4.82
eil101	Reference [47]	696.9	629	10.79
eil101	Reference [48]	689.67	629	9.65
KroA100	This paper	21811	21282	2.49
KroA100	Reference [47]	25398	21282	19.34
KroA100	Reference [49]	21504	21282	1.04
KroB100	This paper	22870	22141	3.29
KroB100	Reference [47]	29002	22141	30.99

The number of synchronizations performed was dependent upon problem size. No attempt was made to find the absolute optimal value for this parameter, but we did increase it with an increase in the number of nodes and in the case of st70 and eil71, this increase was non monotonic. Testing indicated that eight synchronizations worked better for st70 but four worked better for eil71. As a general rule we found that the number of synchronization needed for the framework to perform optimally increased slowly as the size of the problem size increased. The tuning of this parameter is not the principal subject of this work, and is an avenue of possible extensions to this work.

The results clearly justify the application of the parallel migration technique as compared to the primitive parallel approach. Figure 4 Shows the relative performance of the three heuristics is the same across problems, with the migrating parallel outperforming the primitive parallel which outperforms the sequential case.

While our framework was not tuned for the TSP specifically, we show how it performs relative to some published results that were tuned for TSP problems. These results are shown in Table 2. The performance of our framework is competitive with the performance of several other works on the comparison problems. We should note of course that there are some implementations – for example which use a Lin-Kernigan heuristic [46] combined with a GA, that consistently achieve unbelievably good for the TSP even for very large problems. The results shown in Table 2 are merely examples of typical comparable systems. We would like to not that while there are genetic algorithm based solvers for the TSP that outperform our system, these solvers are often highly tuned to the TSP and use heuristics such as Lin-Kernigan which are difficult to generalize to different problems. The framework presented differs on the other hand is almost trivially adaptable to other problem combinatorial problems given the presence of a previously written sequential genetic algorithm solver. In Table 2 we cite results from [47]–[49].

V. CASE STUDY II: MINIMUM MULTIPROCESSOR SCHEDULING PROBLEM

Next we found a fourth year computer science undergraduate student who was willing to do a short term project testing our GA framework and we arbitrarily selected a multiprocessor scheduling problem for that test. GA was not specifically designed for multiprocessor scheduling, but we used such problems to demonstrate how our simple additional solution scaffolding can significantly improve solution quality.

To investigate the gains exhibited by an application of our GA framework would generalize to problems other than the TSP we implemented a generalization of the minimum multiprocessor scheduling (MMS) problem from Gary and Johnson [50]. We used this opportunity to test our claim that a developer with little experience with genetic algorithms and optimization could easily deploy our framework given existing sequential genetic algorithm codes. The student was given a problem description, a brief synopsis of genetic algorithms as they apply to optimization, and code for a genetic algorithm solving the MMS problem sequentially. He was able both to successfully apply the framework to the sequential genetic algorithm code for solving the MMS problem and testing it empirically in a short time. His results show similar gains in using MPB as compared to both PPB and SSB as those described in the earlier TSP results.

A. MMS PROBLEM DESCRIPTION

We are given T tasks $(t_0, \ldots, t_{T-1})$, and P machines $(p_0, \ldots, p_{P-1})$. Each task t when run on any machine will take $Length(t)$ discrete time units to complete. There exists a universal logical clock which will give the logical time in discrete time units at any given instant.

A solution to the MMS problem entails a task to (machine, start time) mapping $M[t_i] = (p_i, t_{i_{start}})$ where t_i is a task from the set Tasks, p_j is a machine from the set machines, and $t_{i_{start}}$ is the time t_i begins to be executed on p_j. The ending time of a task t_i is $t_{i_{start}} + Length(t_i)$. The mapping is constrained as follows: No task may begin execution on a machine at which

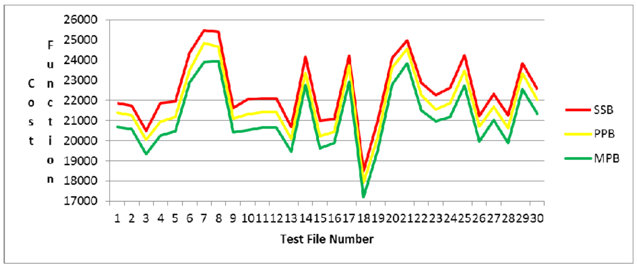

FIGURE 5. Comparison across test cases.

another task is already executing. In other words, given a task t_i beginning execution on machine p_j at time $t_{i_{start}}$ it must be the case that for any other task t_k, task t_k either starts after task t_i finishes or task t_k finishes before task t_i starts:

Assignment $M[t_i] = (p_i, t_{i_{start}})$ is valid only if for all t_k in Tasks such that:

$$\begin{aligned} (k \neq 1) \;\wedge\; (M[t_k] = (p_j, t_{k_{start}})) \\ \rightarrow (t_{k_{start}} \geq (t_{i_{start}} + Length(t_i))) \\ \vee\; ((t_{k_{start}} + Length(t_k)) < t_{i_{start}}) \end{aligned} \tag{4}$$

A mapping M is valid if the above condition holds for all tasks.

Define $EndTime(S_i)$ to be the end time of the latest scheduled task in solution S_i. A solution S_i is considered an optimal solution of a given problem instance if and only if $EndTime(S_i) \leq EndTime(S_j)$, For all alternative solution S_j.

B. TEST CASES

Unlike the empirical tests done on the traveling salesman problem, we did not have a well know library of problems and optimal solutions to choose from, so we generated problem instances randomly. Since we were generating randomized test cases for this problem we were unable to show algorithmic performance as a deviation from the optimal value (since we did not know those). Instead we show the deviation of each solution from the best solutions found (all of which were found by MPB). As before, this is a minimization problem. A group of 30 problem instances was generated, each with a randomly chosen number of machines from within the range [40, 50], a random number of assigned tasks from the range [200, 600], and each task with an arbitrary duration chosen from [1, 2000]. Each of the previously described techniques for solving the problem using a genetic algorithm (SSB, PPB, MPB) was applied to this test set and the results cataloged. For each technique, and every test the results below were averaged over 30 runs of each algorithm on each file. A crossover probability of 80% and a mutation probability of 5% were used. We show these results in Figure 5 and Table 3.

Overall, we found an average improvement of MPB over PPB of 3.25% in terms of solution quality and an average improvement of 6.02% of MPB over SSB. These are averages of 30 runs for 30 randomly generated tests. We believe that these results justify the usefulness of the technique as we were able to achieve an improvement over the primitive design with

TABLE 3. Performance of MPB relative to SSB and PPB.

Test File	Iterations of Internal GA	Migrations	SSB	PPB	MPB	Percent difference from PPB solution to MPB solution (%)
1	4096	12	21884	21385	20692	3.2406
2	4096	12	21726	21271	20581	3.2439
3	4096	12	20511	20038	19347	3.4484
4	4096	12	21869	20952	20274	3.236
5	4096	12	21960	21185	20477	3.342
6	4096	12	24373	23515	22903	2.6026
7	4096	12	25493	24859	23897	3.8698
8	4096	12	25414	24666	23944	2.9271
9	4096	12	21645	21100	20419	3.2275
10	4096	12	22069	21307	20527	3.6608
12	4096	12	22097	21415	20663	3.5116
12	4096	12	22097	21415	20663	3.5116
13	4096	12	20694	20097	19466	3.1398
14	4096	12	24159	23382	22764	2.6431
15	2048	12	20997	20242	19626	3.0432
16	2048	12	21090	20456	19888	2.7767
17	4096	12	24219	23740	22932	3.4035
18	4096	12	18492	17903	17210	3.8709
19	4096	12	21025	20217	19554	3.2794
20	4096	12	24110	23649	22794	3.6154
21	4096	12	24998	24576	23849	2.9582
22	4096	12	22879	22266	21493	3.4717
23	4096	12	22269	21549	20963	2.7194
24	4096	12	22634	21865	21184	3.1146
25	4096	12	24246	23468	22750	3.0595
26	4096	12	21226	20713	19981	3.534
27	4096	12	22315	21681	21029	3.0072
28	4096	12	21252	20638	19902	3.5662
29	4096	12	23859	23342	22563	3.3373
30	4096	12	22570	22024	21350	3.0603

even the most trivial merging and communications functions. Further, a well implemented genetic algorithm will already be generating fairly good results. So, while do not know the optimal solutions in this case, an average these improvements are on top of reasonably good solutions. It seems clear that an even greater improvement would be attained if communications and merging functions relevant to the problem domain were applied. The results clearly show an advantage of using the MPB method as opposed to primitive parallelization as in the PPB approach.

VI. CONCLUSION

This paper presents an intermediate alternative for a genetic algorithm designer between dismissing recent advances in computer hardware or adapting to them very primitively, and re-coding the underlying algorithm to incorporate these advances but at the expense of development time. The parallel GA framework presented at its base is a simple modification of the primitive parallelization concept, but if implemented as described it may be gradually extended to fit the qualities of any underlying problem better. The framework itself does not aim to be a competitor with advanced and rigorous frameworks such as ParadisEO or PISA, it is instead a simple extension of the primitive parallelization technique that improves performance and still leaves room for further incremental extension. The empirical results demonstrate that the presented framework can improve the efficiency of a primitively parallelized genetic algorithm. The framework is not aimed specifically at the optimization problems we used as test cases and can be applied to any candidate combinatorial optimization problem. The empirical results given here, the generality of the approach presented, and the relative ease of implementation of the approach suggest that our parallel GA framework is a preferable option to either primitive parallelization or fine grained parallelization when development time is a binding constraint. This case is further reinforced by the experience of an undergraduate researcher in applying the technique quickly to the minimum multiprocessor scheduling problem.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments. They would also like to thank Jad Aboidiab for testing the minimum multiprocessor scheduling problem.

REFERENCES

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, 2003.

[2] M. Ren, Z. Fan, J. Wu, L. Zhou, and Z. Du, "Design and optimization of underground logistics transportation networks," *IEEE Access*, vol. 7, pp. 83384–83395, 2019.

[3] J. A. Paul and M. Zhang, "Supply location and transportation planning for hurricanes: A two-stage stochastic programming framework," *Eur. J. Oper. Res.*, vol. 274, no. 1, pp. 108–125, Apr. 2019.

[4] M. K. Mehlawat, D. Kannan, P. Gupta, and U. Aggarwal, "Sustainable transportation planning for a three-stage fixed charge multi-objective transportation problem," *Ann. Oper. Res.*, pp. 1–37, 2019.

[5] M. Abbasi, M. Rafiee, M. R. Khosravi, A. Jolfaei, V. G. Menon, and J. M. Koushyar, "An efficient parallel genetic algorithm solution for vehicle routing problem in cloud implementation of the intelligent transportation systems," *J. Cloud Comput.*, vol. 9, no. 1, p. 6, Dec. 2020.

[6] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, "A hybrid genetic algorithm for multidepot and periodic vehicle routing problems," *Oper. Res.*, vol. 60, no. 3, pp. 611–624, Jun. 2012.

[7] M. Allahviranloo, J. Y. J. Chow, and W. W. Recker, "Selective vehicle routing problems under uncertainty without recourse," *Transp. Res. E, Logistics Transp. Rev.*, vol. 62, pp. 68–88, Feb. 2014.

[8] M. Elhoseny, A. Tharwat, and A. E. Hassanien, "Bezier curve based path planning in a dynamic field using modified genetic algorithm," *J. Comput. Sci.*, vol. 25, pp. 339–350, Mar. 2018.

[9] J. Santos, A. Ferreira, and G. Flintsch, "An adaptive hybrid genetic algorithm for pavement management," *Int. J. Pavement Eng.*, vol. 20, no. 3, pp. 266–286, Mar. 2019.

[10] B. Vaissier, J.-P. Pernot, L. Chougrani, and P. Véron, "Genetic-algorithm based framework for lattice support structure optimization in additive manufacturing," *Comput.-Aided Des.*, vol. 110, pp. 11–23, May 2019.

[11] B. Saeidian, M. S. Mesgari, and M. Ghodousi, "Evaluation and comparison of genetic algorithm and bees algorithm for location–allocation of earthquake relief centers," *Int. J. Disaster Risk Reduction*, vol. 15, pp. 94–107, Mar. 2016.

[12] S.-C. Huang, M.-K. Jiau, and C.-H. Lin, "A genetic-algorithm-based approach to solve carpool service problems in cloud computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 1, pp. 352–364, Feb. 2015.

[13] F. Stefanello, L. S. Buriol, M. J. Hirsch, P. M. Pardalos, T. Querido, M. G. C. Resende, and M. Ritt, "On the minimization of traffic congestion in road networks with tolls," *Ann. Oper. Res.*, vol. 249, nos. 1–2, pp. 119–139, Feb. 2017.

[14] K. Hermawan and A. Regan, "On-demand, app-based ride services: A study of emerging ground transportation modes serving Los Angeles international airport (LAX)," *J. Transp. Res. Forum*, vol. 56, no. 3, pp. 111–128, 2017.

[15] K. Hermawan and A. C. Regan, "Impacts on vehicle occupancy and airport curb congestion of transportation network companies at airports," *Transp. Res. Rec.*, vol. 2672, no. 23, pp. 52–58, 2018.

[16] Y. Zhang, L. Bao, S.-H. Yang, M. Welling, and D. Wu, "Localization algorithms for wireless sensor retrieval," *Comput. J.*, vol. 53, no. 10, pp. 1594–1605, 2010.

[17] D. Wu, Y. Zhang, J. Luo, and R. Li, "Efficient data dissemination by crowdsensing in vehicular networks," in *Proc. IEEE Int. Symp. Qual. Service (IWQoS)*, 2014, pp. 314–319.

[18] F. Shi, D. Wu, D. I. Arkhipov, Q. Liu, A. C. Regan, and J. A. McCann, "Parkcrowd: Reliable crowdsensing for aggregation and dissemination of parking space information," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 11, pp. 4032–4044, 2018.

[19] T. D. T. Nguyen, T.-T. Huynh, and H.-A. Pham, "An improved human activity recognition by using genetic algorithm to optimize feature vector," in *Proc. 10th Int. Conf. Knowl. Syst. Eng. (KSE)*, 2018, pp. 123–128.

[20] K. Nitisiri, M. Gen, and H. Ohwada, "A parallel multi-objective genetic algorithm with learning based mutation for railway scheduling," *Comput. Ind. Eng.*, vol. 130, pp. 381–394, Apr. 2019.

[21] D. Wu, L. Lambrinos, T. Przepiorka, D. I. Arkhipov, Q. Liu, A. C. Regan, and J. A. McCann, "Enabling efficient offline mobile access to online social media on urban underground metro systems," *IEEE Trans. Intell. Transp. Syst.*, early access, Apr. 29, 2019, doi: 10.1109/TITS.2019.2911624.

[22] D. Wu, X. Nie, E. Asmare, D. Arkhipov, Z. Qin, R. Li, J. McCann, and K. Li, "Towards distributed SDN: Mobility management and flow scheduling in software defined urban IoT," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1400–1418, 2020.

[23] Z. Tu, R. Li, Y. Li, G. Wang, D. Wu, P. Hui, L. Su, and D. Jin, "Your apps give you away: Distinguishing mobile users by their app usage fingerprints," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 2, no. 3, pp. 1–23, 2018.

[24] A. Lewis and A. Regan, "Enabling paratransit services with blockchain based contracts," in *Proc. Comput. Conf.* London, U.K.: SIA, 2020.

[25] D. Wu, Z. Jiang, X. Xie, X. Wei, W. Yu, and R. Li, "LSTM learning with Bayesian and Gaussian processing for anomaly detection in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 16, no. 8, pp. 5244–5253, 2020.

[26] T. Wu, J. Qin, and Y. Wan, "TOST: A topological semantic model for GPS trajectories inside road networks," *ISPRS Int. J. Geo-Inf.*, vol. 8, p. 410, Sep. 2019.

[27] M. Tang, Z. Li, and G. Tian, "A data-driven-based wavelet support vector approach for passenger flow forecasting of the metropolitan hub," *IEEE Access*, vol. 7, pp. 7176–7183, 2019.

[28] B. Ning, T. Tang, H. Dong, D. Wen, D. Liu, S. Gao, and J. Wang, "An introduction to parallel control and management for high-speed railway systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1473–1483, Dec. 2011.

[29] G. Sun, D. Liao, V. Anand, D. Zhao, and H. Yu, "A new technique for efficient live migration of multiple virtual machines," *Future Gener. Comput. Syst.*, vol. 55, pp. 74–86, Feb. 2016.

[30] D. I. Arkhipov, D. Wu, and A. C. Regan, "A simple genetic algorithm parallelization toolkit (SGAPTk) for transportation planners and logistics managers," pp. 1–18, 2015.

[31] L. Jiacheng and L. Lei, "A hybrid genetic algorithm based on information entropy and game theory," *IEEE Access*, vol. 8, pp. 36602–36611, 2020.

[32] F.-Y. Wang, "Parallel control and management for intelligent transportation systems: Concepts, architectures, and applications," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 3, pp. 630–638, Sep. 2010.

[33] J. Zhang, F.-Y. Wang, K. Wang, W.-H. Lin, X. Xu, and C. Chen, "Data-driven intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1624–1639, Dec. 2011.

[34] J. Lee, K. G. Shin, I. Shin, and A. Easwaran, "Composition of schedulability analyses for real-time multiprocessor systems," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 941–954, Apr. 2015.

[35] N. Hou, F. He, Y. Zhou, Y. Chen, and X. Yan, "A parallel genetic algorithm with dispersion correction for HW/SW partitioning on multi-core CPU and many-core GPU," *IEEE Access*, vol. 6, pp. 883–898, 2018.

[36] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux Syst. Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[37] E. Cantú-Paz and D. E. Goldberg, "On the scalability of parallel genetic algorithms," *Evol. Comput.*, vol. 7, no. 4, pp. 429–449, Dec. 1999.

[38] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee, "Efficient hierarchical parallel genetic algorithms using grid computing," *Future Gener. Comput. Syst.*, vol. 23, no. 4, pp. 658–670, May 2007.

[39] S. Cahon, N. Melab, and E.-G. Talbi, "ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics," *J. Heuristics*, vol. 10, no. 3, pp. 357–380, May 2004.

[40] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA—A platform and programming language independent interface for search algorithms," in *Proc. Int. Conf. Evol. Multi-Criterion Optim.* Springer, 2003, pp. 494–508.

[41] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[42] N. K. Vishnoi, "The speed of evolution," in *Proc. 26th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2015, pp. 1590–1601.

[43] J.-Y. Potvin, "Genetic algorithms for the traveling salesman problem," *Ann. Oper. Res.*, vol. 63, no. 3, pp. 337–370, 1996.

[44] G. A. Sena, D. Megherbi, and G. Isern, "Implementation of a parallel genetic algorithm on a cluster of workstations: Traveling salesman problem, a case study," *Future Gener. Comput. Syst.*, vol. 17, no. 4, pp. 477–488, Jan. 2001.

[45] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, 1991.

[46] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Oper. Res.*, vol. 21, no. 2, pp. 498–516, Apr. 1973.

[47] S. K. Amous, T. Loukil, S. Elaoud, and C. Dhaenens, "A new genetic algorithm applied to the traveling salesman problem," *Int. J. Pure Appl. Math.*, vol. 48, no. 2, pp. 151–166, 2008.

[48] Y. Wei, Y. Hu, and K. Gu, "Parallel search strategies for TSPs using a greedy genetic algorithm," in *Proc. 3rd Int. Conf. Natural Comput. (ICNC)*, 2007, pp. 786–790.

[49] S. S. Ray, S. Bandyopadhyay, and S. K. Pal, "New genetic operators for solving TSP: Application to microarray gene ordering," in *Pattern Recognition and Machine Intelligence*. Berlin, Germany: Springer, 2005, pp. 617–622.

[50] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.

DMITRI I. ARKHIPOV received the B.S. degree in information and computer science, the M.S. degree in computer science, and the Ph.D. degree in computer science from the University of California, Irvine, in 2009, 2012, and 2016 respectively. He is currently a Postdoctoral Researcher with the Department of Computer Science, University of California. His research interests include parallel and distributed systems, large scale combinatorial optimization, and cyber-physical systems.

DI WU (Member, IEEE) received the Ph.D. degree in computer science from the University of California at Irvine, Irvine, CA, USA, in 2013. He was a Researcher with the Intel Collaborative Research Institute for Sustainable Connected Cites, a Research Associate with the Imperial College London, a Staff Research Associate with the University of California at Irvine, a Visiting Researcher at IBM Research, and a Student Research Associate at SRI International. He is currently a Professor with Hunan University, China, and an Adjunct Researcher with the University of California at Irvine. His research interests include future networking, intelligent analytics, and smart architecture. He has actively served on many conference committees. He is an Associate Editor of the IEEE Transactions on Intelligent Transportation Systems.

TAO WU received the B.S. degree in resources environment and the management of urban and rural planning from the Wuhan University of Technology, Wuhan, China, the M.S. degree in cartography and geography information system from Central South University, Changsha, China, and the Ph.D. degree in geographic information science from the Joint Doctoral Program between Wuhan University and Central South University. He is currently working as a Postdoctoral Researcher with the Key Laboratory of Geospatial Big Data Mining and Application, Changsha. His research interests include geographic information systems, large scale spatio-temporal trajectory data mining, and smart cities.

AMELIA C. REGAN (Member, IEEE) received the B.A.S. degree in systems engineering from the University of Pennsylvania, the M.S. degree in applied mathematics from Johns Hopkins University, and the M.S. and Ph.D. degrees in transportation systems engineering from The University of Texas at Austin, Austin. Prior to receive the Ph.D. degree, she was an Operations Research Analyst with the Association of American Railroads and United Parcel Service. She has also taught short courses at the Athens University of Business and Economics and the National Technical University of Denmark. She is currently a Professor of computer science and transportation systems engineering with the University of California, Irvine. Her research is focused on algorithm development for optimization of transportation and communication systems.