Using a Parallel Ensemble of Sequence-Based Selection Hyper-Heuristics for Electric Bus Scheduling

Darren M. Chitty
Faculty of Environment, Science and Economy
University of Exeter, UK
darrenchitty@googlemail.com

James Lewis
City Science
Exeter, UK
james.lewis@cityscience.com

Ed Keedwell
Faculty of Environment, Science and Economy
University of Exeter, UK
E.C.Keedwell@exeter.ac.uk

ABSTRACT

A Sequence-based Selection Hyper-Heuristic (SSHH) utilises a hidden Markov model (HMM) to generate sequences of low-level heuristics to apply to a given problem. The HMM represents learnt probabilistic relationships in transitioning from one heuristic to the next for generating good sequences. However, a single HMM will only represent one learnt behaviour pattern which may not be ideal. Furthermore, using a single HMM to generate sequences is sequential in manner but most processors are parallel in nature. Consequently, this paper proposes that the effectiveness and speed of SSHH can be improved by using multiple SSHH, an ensemble. These will be able to operate in parallel exploiting multi-core processor resources facilitating faster optimisation. Two methods of parallel ensemble SSHH are investigated, sharing the best found solution amongst SSHH instantiations or combining HMM information between SSHH models. The effectiveness of the methods are assessed using a real-world electric bus scheduling optimisation problem. Sharing best found solutions between ensembles of SSHH models that have differing sequence behaviours significantly improved upon sequential SSHH results with much lower run-times.

CCS CONCEPTS

• **Computing methodologies** → **Search methodologies**.

KEYWORDS

hyper-heuristics, parallelism, ensemble optimisation

1 INTRODUCTION

For complex combinatorial optimisation problems there are a number of approaches that can be used, simple heuristics, local search methods or meta-heuristics such as a Genetic Algorithm (GA) [16]. A common methodology is a GA combined with local search. However, in recent years a technique that can be used to solve problems from multiple domains without tailoring is a hyper-heuristics approach. Indeed, a hyper-heuristic can be used to select low-level heuristics to apply to a solution to improve it or even configure meta-heuristics themselves. Hyper-heuristics can be relatively simple in nature employing a simple random or greedy method or more advanced such as generating sequences of low-level heuristics to apply sequentially such as Sequence-Based Selection Hyper-Heuristics (SSHH) [18].

ISBN 979-8-4007-0120-7/23/07
https://doi.org/10.1145/3583133.3596340

Hyper-heuristics such as SSHH typically operate in a sequential manner but most processors are parallel in nature with multiple cores such that algorithms should equally be parallel in nature. This is a key advantage for meta-heuristics many of which are population-based and hence easily parallelisable increasing their speed and ability to optimise large problems. Therefore, it can be concluded that hyper-heuristics themselves also need to be parallel in operation to fully exploit multi-core processor resources. Of further consideration is that SSHH uses a singular hidden Markov model (HMM) to generate sequences which is a single learnt model. However, a set or *ensemble* of HMMs may provide improved accuracy in terms of optimisation and an easy route to parallelisation. Consequently, this paper proposes to assess a parallel ensemble of SSHH models using a large real-world electric bus scheduling problem. The paper is laid out as follows: Section 2 profiles hyper-heuristics and associated parallel implementations. Section 3 introduces SSHH and two parallel ensemble models that could be used to improve both the speed and the accuracy of the optimisation. Section 4 uses a real-world electric bus scheduling problem to test and compare SSHH and the two parallel methodologies. Finally Section 5 summarises the work and future research directions.

2 BACKGROUND AND RELATED WORK

2.1 Hyper-Heuristics

An approach that has recently gained ground in combinatorial optimisation is a hyper-heuristic methodology [7]. Whereas meta-heuristics search within the space of problem solutions, a hyper-heuristic searches within the space of heuristics that operate within the solution space to find an improvement [3]. The hyper-heuristic does not use problem domain knowledge within its operation. The low-level heuristics utilised range from methods that are simple such as swapping two vertices in a solution to performing a full local search methodology such as 2-opt [8]. Hyper-heuristics can be generally categorised into two groups, selection hyper-heuristics which apply a low-level heuristic at each iteration and generational hyper-heuristics which create novel low-level heuristics.

Cowling et al. [7] considered a range of simple hyper-heuristic methods to select heuristics. For instance, Simple Random (SR) which uses uniform probability of selection, Random Descent (RD) which is similar to SR but continues to use the same heuristic until no longer successful, and two permutation methods Random Permutation and Random Permutation Descent which iterate through a random permutation of heuristics similar to SR and RD respectively. A Greedy method (GR) applies all low-level heuristics and accepts the heuristic with the best result.

More advanced hyper-heuristics consider the application of *sequences* of heuristics. In this instance more than one low level heuristic is applied sequentially to the current solution before testing the resulting solution for an improvement. A sequence can be beneficial as a number of changes might need to be applied to break a solution out of a local optima. Iterated Local Search (ILS) [20] was an early attempt to create sequences of low-level mutation heuristics and then rebuild with local search heuristics [21]. AdapHH is a state of the art hyper-heuristic which considered the pairings of destructive and constructive low-level heuristics including GA crossover operators [23]. An Evolutionary Programming Hyper-heuristic (EPH) attempts a population-based methodology whereby a population of low-level heuristic sequences is maintained using diversification and intensification operations [22]. An alternative hyper-heuristic methodology for generating sequences uses a Monte Carlo tree search whereby the low-level heuristics are represented as a tree and searched to find the best sequence to apply to the current solution [29]. The Sequence-based Selection Hyper-Heuristic (SSHH) generates sequences of heuristics by attempting to learn relationships between heuristics. [18]. This is achieved by using a hidden Markov model (HMM) to represent the probability of using one heuristic after another has been previously applied. Each heuristic use is considered as a *state* and the HMM models the transitional probabilities of moving from one state to the next. The success of a heuristic sequence is measured and if an improvement is found, the success count for each transition in the sequence is incremented. In terms of the sequence length, a second HMM is used after each heuristic to determine if the sequence should end or continue. Drake et. al provide a more complete recent overview of advances in selection hyper-heuristics [11].

2.2 Parallel Hyper-Heuristics

Parallel multi-core architectures are now commonplace, modern CPUs have eight to 32 processor cores available for use. Furthermore, a high performance processing architecture known as a Graphical Process Unit (GPU) is commonly found in computing platforms which have thousands of processor cores available. Therefore, computationally heavy tasks such combinatorial optimisation need to make use of these parallel architectures. Indeed, most meta-heuristics have long exploited parallel processor resources often due to their population-based nature being easy to parallelise and achieve significant speedups. For instance, a GA can be implemented in parallel simply by having each processor core evaluate a population member [14]. Or alternatively, a number of GA sub-populations can be implemented in parallel with migration occurring between these parallel populations termed an island model [32]. GPU implementations of GAs have also been implemented, Vidal and Alba [34] implemented a cellular GA on a GPU reporting a speedup of up to 25 fold when using very large population sizes in the order of 25,000. The island model for a GA can also be implemented upon a GPU with Pospichal et al. [26] implementing an island GA on a GPU with asynchronous migration reporting speedups of up to 8000 fold for numerical optimisation

Consequently, hyper-heuristics should also operate in a parallel manner to achieve best performance for combinatorial optimisation problems but are generally sequential in nature. However, a limited form of hyper-heuristic parallelism has been achieved albeit in a master-slave manner. The hyper-heuristic acts a controller or master selecting parallel meta-heuristics to be used and selecting their associated parameters. The parameters for parallel implementations of a GA such as an island model can be more complicated due to specifying parameters such as the sub-population sizes and the migration rates. For example León et al. [19] used a hyper-heuristic to deploy parallel island meta-heuristics with the hyper-heuristic granting more computational resources to promising meta-heuristics. Dokeroglu and Cosar [9] use similar approach with a master-slave model whereby the hyper-heuristic controls the meta-heuristics sending solutions to be evaluated in parallel. In a second phase the best meta-heuristic is implemented on every node in parallel. Segura et al. [30] use a hyper-heuristic with a parallel island model GA whereby the hyper-heuristic controls the configurations of each island. Rodriguez et al. [28] exploit parallelism by using three differing meta-heuristics, a GA, Ant Colony Optimisation (ACO) [10] and Simulated Annealing allowing a hyper heuristic to select a number of each to operate in parallel. Bertels et al. [2] demonstrated that an asynchronous parallel population based evolutionary algorithm could speedup the design of Boolean satisfiability solvers significantly. Oteiza et al. [25] use a similar approach with a GA, Simulated Annealing and Particle Swarm Optimisation (PSO) [12]. Imbernón et al. [17] use GPU clusters with a hyper-heuristic selecting which parallel meta-heuristics to use on the GPU cluster at each step. Alekseeva et al. [1] use a master-slave hyper-heuristic approach to find the best GRASP setup with multiple configurations evaluated on differing cores.

This master-slave parallel hyper-heuristic method is easy to implement such that an extension to HyFlex [24], ParHyFlex, enables users to test hyper-heuristics in parallel [33]. Rattadilok et al. [27] though do consider the use of a hyper-heuristic in parallel whereby a choice function hyper-heuristic method executes multiple low-level heuristics in parallel to effectively speedup the optimisation search process. A hierarchical scheme is used where the hyper-heuristic sends low-level heuristics to CPU cores which iteratively apply them until no improvements whereby solutions are sent to the controlling hyper-heuristic to use or discard. This is extended to have multiple hyper-heuristic agents operating in parallel.

3 PARALLEL SEQUENCE-BASED SELECTION HYPER-HEURISTIC MODELS

The Sequence-based Selection Hyper-Heuristic (SSHH) [18] constructs sequences using a hidden Markov model (HMM) whereby low-level heuristics are considered states and the HMM provides a probability of transitioning from one heuristic to the next. If a set of n low-level heuristics is defined as $[llh_0, llh_1, ...llh_{n-1}]$ then the transition probabilities of moving from one heuristic to the next is defined by an n by n matrix. The SSHH transition probabilities are defined by the success counts of heuristic transitions having occurred in sequences which improved upon the current best solution. SSHH also uses a sequence acceptance HMM to decide after a heuristic selection if the sequence should be accepted or to continue construction, an n by 2 matrix. Algorithm 1 provides a top-level overview of the SSHH process. The selection of a low-level heuristic occurs on line 7 whereby a roulette wheel selection is

made using the HMM transition probabilities $Trans$ from the current last used heuristic $curr$, $Tran[curr][next]/\sum_{\forall j} Tran[curr][j]$.

Algorithm 1 SSHH

```
1: S, S', S_b = candidate, new and best solutions respectively
2: Tran, Seq = the transition and sequence matrices
3: [llh_0, llh_1, llh_2, ..., llh_{n-1}] = the low-level heuristics
4: HeuristicSequence is the current heuristic sequence
5: curr = select random low-level heuristic
6: while iteration less than max iterations do
7:     next=SelectNext(Tran,curr)
8:     Add toHeuristicSequence next
9:     AcceptStatus=ComputeStatus(Seq,next)
10:    if Status = complete sequence then
11:        S' = application of HeuristicSequence
12:        if S' better than S_b then
13:            S_b=S'
14:            Update Tran and Seq success counts
15:        end if
16:        S=S'
17:        Clear HeuristicSequence
18:    end if
19:    curr=next
20: end while
```

However, this operation of SSHH is sequential in manner, a single set of transition and acceptance HMMs generating a sequence of low-level heuristics which are applied to a singular best solution. Contrasting to popular meta-heuristics such as GAs or ACO these methods are population-based and hence there are many parallel implementations of these approaches. A simple method of parallelisation of a GA is to evaluate the population of solutions in parallel [4] or facilitate whole sub-populations in parallel across multiple processor cores with migration of individuals between cores [31]. ACO is even more parallelisable as ants constructing solutions themselves is computationally expensive but ants only communicate through a pheromone matrix such that they are relatively independent. Consequently, for SSHH to maintain parity with meta-heuristic approaches it should also be parallel in nature.

Of further consideration is the behaviour of SSHH itself with just a single transition and acceptance HMM and a single best found solution for heuristics in a sequence to act upon. This could impede the performance of SSHH as the learnt transitions in the HMM may not be ideal. Indeed, the transitions learnt in the early stages of the optimisation process may be less relevant in the later stages but there is just a single HMM model and changing the transition probabilities relies on finding new best solutions. However, in the field of classification, performance of a classifier improves when using a collection of learnt models, an ensemble approach. This approach has been used with multiple decision trees forming a random forest or multiple neural networks. Indeed, an ensemble of classifiers is considered more accurate as long as they are diverse [15]. Of further consideration is SSHH using a single solution to operate heuristic sequences upon. If there are multiple solutions, a population effectively, this may enable local optima to be escaped through diversity of solutions. The advantage of using multiple instantiations of SSHH in parallel, an ensemble, is that there should be no additional computational cost. Indeed, if less iterations are necessary through using parallel SSHH implementations there could also be an effective speedup. Ensemble optimisation has been used previously to great effect combining a GA, PSO and social spider optimisation [13].

Consequently, it is hypothesised that operating multiple parallel implementations of SSHH will both improve the effectiveness of the approach and the convergence speed. Clearly, a simple methodology would be to just execute multiple instantiations of SSHH in parallel and then return the best found solution over all parallel SSHH runs. This is the same as running SSHH for multiple seeded runs and returning the best result. Clearly this would lead to an improvement in the results from SSHH with no additional computational cost. However, it is also hypothesised that sharing information between parallel SSHH models during the optimisation process would be more effective. Sharing of information between multiple parallel SSHH models can be achieved in two ways, either sharing of the best found solution so far amongst the multiple SSHH models or alternatively the HMM transition probabilities from each SSHH model.

3.1 Shared SSHH Transition Probabilities

With this parallel SSHH model a set of separate instantiations of SSHH are initiated whereby each SSHH model has its own transition and sequence acceptance HMMs and its own best found solution. The parallel SSHH models conduct the standard SSHH model as described in Algorithm 1 updating transition probabilities of their HMMs if a solution derived by a sequence is an improvement on their own best found solution. This process is conducted in an asynchronous manner with no communication or waiting for other parallel SSHH models to reach the same iteration. However, after a set number of these asynchronous iterations the parallel SSHH models communicate their HMM transition probabilities, a synchronisation step whereby parallel threads need to wait until all have reached the same iteration. This model will be termed ParallelSSHH-SharedHMM.

To combine the HMMs of all the parallel SSHH models is straightforward. The transitional probabilities of the HMMs in SSHH are simply the success counts from when a transition from one state to another occurs in a sequence which improves upon their locally held best solution. Consequently, for each transitional state in the HMMs the success counts can be summed. This forms a new *global* transitional HMM and sequence acceptance HMM which when complete are copied over all the local parallel HMMs such that all threads will have at their next iteration the same HMM behaviour. Obviously, with each parallel SSHH model having differing best found solutions the parallel HMMs will become diverse until the next synchronisation step. The diversity of the parallel SSHH model is maintained with each parallel SSHH model having their own best found solution.

The operation of the parallel shared SSHH transition probabilities model is shown in Algorithm 2. Note that lines 2 to 8 now set up SSHH for multiple parallel implementations. There is also now global transition and sequence matrices set up on line 1. Also, note the additional loops on lines 10 and 11 that run each parallel SSHH model i for a fixed number of asynchronous iterations. The synchronisation step and combining of SSHH transition probabilities occurs from line 27 onwards whereby for each transition probability the total success counts are summed across all parallel SSHH models. A record of the new global HMMs are kept in

Algorithm 2 Parallel SSHH Sharing HMM State Transition Success Counts

```
1: GlobalTran, GlobalSeq = the global transition and sequence matrices
2: for each parallel SSHH implementation i do
3:     S_i, S'_i, S_ib = candidate, new and best solutions respectively
4:     Tran_i, Seq_i = the transition and sequence matrices
5:     [llh_0, llh_1, llh_2, ..., llh_{n-1}] = the low-level heuristics
6:     HeuristicSequence_i is the current heuristic sequence
7:     curr_i = select random low-level heuristic
8: end for
9: while total iterations less than max iterations do
10:    for each parallel SSHH implementation i do
11:        for each asynchronous iteration do
12:            next_i=SelectNext(Tran_i,curr_i)
13:            Add to HeuristicSequence_i next_i
14:            AcceptStatus_i=ComputeStatus(Seq_i,next_i)
15:            if Status_i = complete sequence then
16:                S'_i = application of HeuristicSequence_i
17:                if S'_i better than S_ib then
18:                    S_ib=S'_i
19:                    Update Tran_i and Seq_i success counts
20:                end if
21:                S_i=S'_i
22:                Clear HeuristicSequence_i
23:            end if
24:            curr_i=next_i
25:        end for
26:    end for
27:    for each transition state t do
28:        Sum=0
29:        for each parallel SSHH implementation i do
30:            Sum+=Tran_it
31:        end for
32:        for each parallel SSHH implementation i do
33:            Tran_it=Sum-GlobalTran_t
34:        end for
35:        GlobalTran_t=Sum-GlobalTran_t
36:    end for
37:    for each sequence state s do
38:        Sum=0
39:        for each parallel SSHH implementation i do
40:            Sum+=Seq_is
41:        end for
42:        for each parallel SSHH implementation i do
43:            Seq_is=Sum-GlobalSeq_s
44:        end for
45:        GlobalSeq_s=Sum-GlobalSeq_s
46:    end for
47: end while
```

GlobalTrans and *GlobalSeq* and these are used to measure the increase in transition success counts between synchronisation steps.

3.2 Shared SSHH Best Found Solutions

This proposed parallel SSHH model is similar to the prior model, ParallelSSHH-SharedHMM. Once again, a set of parallel instantiations of SSHH are initiated using their own heuristic transition and sequence acceptance HMMs and local best found solutions. Their process operates the same as described in Algorithm 1 during their asynchronous iterations whereby there is no communication between parallel SSHH models. However, after their set of asynchronous iterations the parallel SSHH models communicate their best found solutions so far to the master thread, the synchronisation step whereby all parallel SSHH models reach the same iteration. This parallel SSHH model will be termed ParallelSSHH-SharedBest.

To communicate the best found solution from all SSHH models back to parallel SSHH instantiations, the master thread simply identifies the best solution from all the local best found solutions. This is then copied over to each parallel thread overwriting their locally held best found solution. Thus at this point all parallel SSHH instantiations have the same best found solution to apply their locally generated sequences to. The diversity of this parallel model is achieved with each parallel SSHH model having differing behaviours from their HMMs.

Algorithm 3 Parallel SSHH Sharing Best Found Solution Between Models

```
1: GlobalBest = the globally best found solution
2: for each parallel SSHH implementation i do
3:     S_i, S'_i, S_ib = candidate, new and best solutions respectively
4:     Tran_i, Seq_i = the transition and sequence matrices
5:     [llh_0, llh1_1, llh_2, ..., llh_{n-1}] = the low-level heuristics
6:     HeuristicSequence_i is the current heuristic sequence
7:     curr_i = select random low-level heuristic
8: end for
9: while total iterations less than max iterations do
10:    for each parallel SSHH implementation i do
11:        for each asynchronous iteration do
12:            next_i=SelectNext(Tran_i,curr_i)
13:            Add to HeuristicSequence_i next_i
14:            AcceptStatus_i=ComputeStatus(Seq_i,next_i)
15:            if Status_i = complete sequence then
16:                S'_i = application of HeuristicSequence_i
17:                if S'_i better than S_ib then
18:                    S_ib=S'_i
19:                    Update Tran_i and Seq_i success counts
20:                end if
21:                S_i=S'_i
22:                Clear HeuristicSequence_i
23:            end if
24:            curr_i=next_i
25:        end for
26:    end for
27:    for each parallel SSHH implementation i do
28:        if S'_ib is better than GlobalBest then
29:            GlobalBest=S'_ib
30:        end if
31:    end for
32:    for each parallel SSHH implementation i do
33:        S'_ib=GlobalBest
34:        S_i=GlobalBest
35:    end for
36: end while
```

The operation of ParallelSSHH-SharedBest is shown in Algorithm 3. This is similar to Algorithm 2 but on line 1 there is now a global track of the best found solution updated at each synchronisation step. Again, synchronisation occurs from line 27 whereby for each parallel instantiation of SSHH the best solution is tested against the global best updating if better. Once this is achieved the global best is communicated back to all the parallel SSHH models for them to use before the next asynchronous stage begins.

4 EXPERIMENTAL RESULTS

4.1 Real-World Electric Bus Scheduling

To measure the effectiveness of the parallel ensemble SSHH models vs. the sequential standard SSHH a difficult combinatorial real-world problem will be used, the optimal scheduling of electric buses. The goal of electric bus scheduling is to service all the timetabled stops with no tardiness whilst minimising the size of the electric bus fleet required and total fleet distance traversed. A solution

Table 1: Real-world electric bus routing problems

Problem	Geographic Lines	Total Line Trips	Available Buses	Total Trip Distance (km)
Scenario A	20	890	150	13555.99
Scenario B	20	518	150	12145.38
Scenario C	24	670	150	7244.83
Scenario D	60	1456	300	21050.11
Scenario E	64	1774	300	29836.56
Scenario F	124	3230	500	50886.67

which uses fewer buses is considered better than a solution which uses more buses but a lower traversal distance. In addition to the timetable constraint, with electric buses an additional constraint exists in that the buses have a fixed range. Therefore, the schedule must ensure that each electric bus can service its assigned timetabled trips and return to the depot without running out of charge otherwise significant tardiness would occur. For the real-world problem a UK bus operator operates buses throughout a large area with a radius of 50 km. Buses run on a range of routes operating to a given timetable. A fleet of electric buses are used equipped with a 450kWh battery providing a range of 185 km using 2.42kWh of energy per km. A candidate solution will consist of a set of unique values representing buses each followed by unique values representing its assigned timetabled trips. An electric bus performs these trips in their assigned order. A set of routing scenarios have been created from the UK bus operator as described in Table 1.

To apply the SSHH hyper-heuristic methods to the electric bus scheduling problem a range of low-level heuristics must be made available. For this problem six low-level heuristics are used:

Swap: Selects two random trips assigned to differing electric buses and exchanges these two trips in their schedules.

Insert: Selects random trip assigned to a bus and and inserts into a random position in a second electric bus schedule.

Invert: Randomly selects two points within a bus fleet schedule and reverses all the trips between the two points.

Reconstruction: Advanced heuristic which randomly selects up to 30 buses operating in similar geographical area and rebuilds their schedules using a probabilistic model based on minimising non-service time lost [5, 6].

Local Search Swap: Two electric buses are selected and every bus trip in each iteratively swapped with improvements retained.

Local Search Insert: Two electric buses are selected and each trip in second bus schedule is iteratively inserted into every slot in the first bus schedule with improvements retained.

4.2 Comparing SSHH and Parallel Multi-SSHH

To begin with standard SSHH and a parallel implementation of SSHH whereby each parallel instantiation executes asynchronously with no sharing between threads will be tested (Multi-SSHH). This is the equivalent of simply running SSHH multiple times and reporting the best result. Due to the high degree of complexity of the electric bus scheduling problem a large degree of iterations are used. Indeed, SSHH will generate 5 million sequences of up to a maximum length of 10 heuristics with improvements greedily accepted. With the parallel implementations the same number of sequences will be generated so if there are 10 parallel instantiations of SSHH then each will iterate for just 500,000 iterations. Experiments will use a Ryzen 2700 processor which has eight cores but via hyper-threading can execute up to sixteen threads simultaneously thus an ensemble of sixteen parallel instantiations of SSHH will be used. Experiments are conducted over 25 random runs.

The results from comparing standard SSHH and Multi-SSHH are shown in Table 2 in terms of the non-service distance travelled, the only distance that can be minimised, the electric bus fleet size, the execution timings and the average length of sequences generated. These results show that whilst SSHH appears to minimise the fleet traversal distance better it does so consistently using more buses than the solutions from Multi-SSHH. In terms of solution quality, using fewer electric buses is considered the best option. It is unclear why the traversal distances are much larger for Multi-SSHH but reduced iterations is the likely cause. Regards execution time Multi-SSHH is significantly faster by approximately seven fold. A key reason is the reduction in iterations used in Multi-SSHH to ensure that the same number of sequences, five million, are generated for a fair comparison of solution quality. Sequence lengths are comparable between the two implementations of SSHH.

4.3 Sharing HMM Information Between Parallel SSHH Models

The next step is to consider the sharing of information between parallel SSHH implementations. As described in Section 3 after a set number of asynchronous iterations information needs to be exchanged between the parallel SSHH implementations, a synchronisation step. In this case each parallel SSHH model communicates its HMM transition probabilities in the form of the success counts. The master thread collates these and generates a global heuristic and sequence acceptance HMM and send this back to each parallel SSHH model to use. A range of asynchronous iterations between HMM sharing steps will be tested to assess the best level to use.

The results from these experiments are shown in Table 3 where it can firstly be observed that for the non-service distances better results are achieved when the number of asynchronous generations is a high number but in terms of the bus fleet size the number of buses increases as the degree of sharing reduces. Comparing to Table 2 these results are considerably poorer even than standard sequential SSHH. A reason for this is that with multiple seeded runs and standard sequential SSHH, sometimes poor results occur which is due to the heuristics and sequence acceptance HMM developing poor transition probabilities that cannot be reversed leading to low quality sequences and results (note the high degree of variance). Therefore, it can be postulated that if one parallel SSHH HMM develops a set of poor transition probabilities when these are combined this *contamination* could cause the new global HMMs to be equally bad. In terms of the runtimes note that with a low-level of asynchronous iterations these are much higher than when using a large degree of asynchronous iterations. For smaller problems, 10,000 asynchronous iterations is several fold faster but with

Table 2: Average optimised fleet non-service distance, bus fleet sizes, runtimes and sequence lengths when using standard sequential SSHH and parallel Multi-SSHH for each bus routing scenario.

Scenario	Average Non-Service Distance Travelled (km)		Average Bus Fleet Size		Average Execution Time (s)		Average Sequence Length	
	SSHH	Multi-SSHH	SSHH	Multi-SSHH	SSHH	Multi-SSHH	SSHH	Multi-SSHH
A	3300.98±351.37	**3173.41±101.37**	102.40±10.14	**97.52±1.00**	373.90±156.41	53.04±6.44	3.99±2.76	3.91±0.67
B	**2133.24±133.36**	2135.47±65.96	84.04±3.31	**82.64±0.76**	199.41±86.49	29.31±4.16	3.74±2.82	4.05±0.65
C	**1107.49±91.69**	1183.21±75.83	49.28±1.77	**49.16±0.47**	630.09±400.31	91.82±6.45	3.83±2.56	3.32±0.63
D	**5987.32±476.20**	6240.71±266.13	**160.64±5.48**	160.96±1.43	543.41±143.43	76.95±9.72	4.96±2.75	3.73±0.67
E	**5082.30±447.24**	5495.26±207.82	207.48±18.86	**204.48±1.71**	605.12±208.79	87.16±11.89	3.25±2.60	4.02±0.64
F	**11 754.19±1084.71**	13 740.09±1449.41	397.52±47.09	**377.68±7.22**	792.12±347.29	168.91±55.29	2.84±2.59	3.05±0.63

Table 3: Average optimised fleet non-service distance, bus fleet sizes, runtimes and sequence lengths when combining HMMs using a range of asynchronous iterations between parallel sharing of HMMs

	Asynchronous Iterations	Electric Bus Fleet Routing Scenario					
		A	B	C	D	E	F
Non-Service Distance (Km)	5	**3342.04±217.50**	2132.75±74.43	1172.84±94.32	6594.31±310.14	5941.15±484.51	14 261.54±1638.05
	10	3385.62±194.77	2151.34±80.36	1193.85±95.72	6514.87±222.21	5900.98±423.67	13 857.65±1992.43
	100	3384.56±144.52	**2110.84±55.01**	1154.98±66.33	6470.20±243.45	5694.55±341.62	12 606.01±906.44
	1000	3342.64±93.03	2123.81±58.05	**1151.55±61.18**	6533.77±322.88	5376.41±204.04	12 376.34±835.47
	10000	3398.84±67.85	2117.45±70.74	1210.97±66.84	**6400.70±159.69**	**5350.66±124.64**	**12 059.86±522.44**
Bus Fleet Size	5	**100.52±4.43**	**82.60±1.26**	**49.56±1.04**	164.88±5.54	**212.12±11.90**	404.76±32.10
	10	101.24±3.92	83.16±1.25	49.96±1.72	165.84±5.71	214.92±10.92	406.88±39.82
	100	102.12±2.65	83.40±0.76	49.92±0.76	**164.60±3.18**	214.92±5.20	**395.40±26.91**
	1000	101.36±1.87	83.32±0.69	50.36±0.70	169.00±2.81	213.72±3.51	397.24±15.71
	10000	102.92±1.58	83.72±0.68	51.36±1.04	171.20±4.14	219.24±4.00	409.92±16.24
Execution Time (s)	5	104.33±28.48	74.54±4.95	148.98±52.12	130.75±25.11	133.95±22.36	176.05±63.32
	10	78.72±21.13	54.05±8.65	124.80±54.49	125.26±40.97	110.72±26.58	161.75±55.38
	100	66.63±10.58	37.02±5.66	149.53±26.99	89.17±11.20	99.99±15.65	141.60±39.33
	1000	65.71±5.26	33.03±3.56	129.73±24.68	75.79±4.35	94.53±6.12	125.27±28.29
	10000	56.57±2.33	28.82±1.98	108.47±19.93	72.60±3.55	86.55±6.18	115.97±26.08
Sequence Length	5	3.68±3.68	3.14±3.14	4.04±2.72	4.43±2.62	3.74±2.77	2.68±2.12
	10	4.32±4.32	4.48±4.48	4.16±2.70	5.14±2.50	4.17±2.82	3.57±2.11
	100	6.57±6.57	6.60±6.60	6.22±1.25	6.17±1.85	7.13±1.25	5.53±2.21
	1000	7.05±7.05	6.71±6.71	6.36±1.07	7.35±0.12	7.41±0.05	6.73±1.31
	10000	6.49±6.49	6.32±6.32	6.15±0.53	6.89±0.23	6.97±0.11	6.20±1.06

Table 4: Average fleet non-service distance, sizes, runtimes and sequence lengths when combining HMMs which only update success scores if new solution better than global best solution using a range of asynchronous iterations between parallel shares

	Asynchronous Iterations	Electric Bus Fleet Routing Scenario					
		A	B	C	D	E	F
Non-Service Distance (Km)	5	3326.61±131.03	2163.78±98.38	1188.37±90.67	6569.20±295.50	5749.26±314.13	14 152.38±1842.35
	10	3338.75±155.02	2139.71±80.08	1180.02±70.27	6695.50±336.34	5953.39±428.83	13 444.03±1137.13
	100	3320.11±113.48	**2094.40±48.63**	1136.42±42.38	6359.52±258.80	5493.64±278.17	12 373.57±715.76
	1000	**3209.28±65.54**	2097.93±57.71	**1118.61±40.51**	**6126.90±195.66**	5194.06±166.33	**11 874.06±653.12**
	10000	3215.38±47.73	2106.36±67.31	1169.52±46.74	6138.05±138.89	**5189.99±137.30**	12 093.90±510.49
Bus Fleet Size	5	**99.76±2.26**	82.96±1.49	49.88±2.09	165.48±5.53	210.80±4.98	407.00±37.31
	10	99.80±2.50	83.08±0.86	49.64±0.95	164.96±3.65	211.68±5.47	402.48±31.46
	100	100.04±2.19	82.84±0.62	49.72±0.68	161.88±1.33	206.96±2.61	383.16±12.29
	1000	98.80±1.32	**82.52±0.51**	**49.20±0.58**	**160.76±0.97**	**203.32±1.25**	**374.72±6.11**
	10000	98.88±0.83	82.56±0.51	49.60±0.58	161.72±1.06	204.40±0.71	383.64±7.50
Execution Time (s)	5	97.53±19.65	73.25±3.90	140.46±49.17	121.87±10.77	132.71±18.57	173.69±60.25
	10	75.35±20.82	51.19±4.30	121.49±47.18	109.38±22.24	104.26±23.95	160.86±45.03
	100	72.06±11.05	36.15±6.05	142.26±35.21	96.25±17.13	109.84±18.20	155.56±27.68
	1000	69.68±8.37	35.91±4.09	129.68±20.98	93.01±10.31	114.45±13.58	154.28±40.76
	10000	60.78±6.05	31.71±2.81	100.52±17.56	82.97±6.70	100.04±8.02	123.67±24.27
Sequence Length	5	3.57±3.57	3.46±3.46	3.65±2.44	4.51±2.40	4.45±2.66	3.23±2.64
	10	3.03±3.03	3.68±3.68	3.70±2.34	4.04±2.54	3.66±2.70	3.62±2.48
	100	5.93±5.93	5.25±5.25	5.23±1.48	5.32±1.87	6.29±1.36	5.00±1.84
	1000	6.09±6.09	5.57±5.57	4.94±0.96	5.95±0.79	6.09±0.76	5.26±1.24
	10000	5.25±5.25	5.10±5.10	4.64±0.68	5.75±0.53	6.14±0.43	4.97±1.07

larger problems the speedup is lower. The reason is that at the synchronisation step threads must wait for all to catch up as some are slower as a result of using more intensive heuristics or longer sequences. This reduces the CPU occupancy which will increase the runtime, the more synchronisation steps the lower the CPU occupancy. Note average sequence size tends to be longer with greater numbers of asynchronous iterations such that the average computational cost per heuristic executed is further improved. Also note that when sharing HMM information the execution times for the larger problems are lower than for Multi-SSHH with no sharing and complete asynchronous iterations. This is due to Multi-SSHH waiting until the last SSHH implementation completes.

Table 5: Average optimised fleet non-service distance, bus fleet sizes, runtimes and sequence lengths when sharing the best solution across parallel threads using a range of asynchronous iterations between parallel shares

	Asynchronous Iterations	Electric Bus Fleet Routing Scenario					
		A	B	C	D	E	F
Non-Service Distance (Km)	5	**3040.33±96.30**†	2038.28±69.11†	1048.12±58.40†	**5444.53±116.24**†	4916.97±135.52	10 802.49±330.72†
	10	3072.37±73.96†	2051.06±63.85†	1040.87±74.65†	5464.44±116.19†	**4864.93±118.58**†	10 731.81±358.38†
	100	3060.50±75.50†	2036.41±64.27†	**1030.71±58.70**†	5560.42±119.41†	4891.03±83.00†	**10 620.72±215.93**†
	1000	3087.87±92.72†	2050.88±57.24†	1049.17±55.47†	5688.91±100.60†	4968.05±117.94	11 056.53±262.09†
	10000	3141.80±66.85	**2033.21±63.16**†	1083.14±57.30	5949.61±115.96	5155.17±100.90	12 110.57±389.37
Bus Fleet Size	5	**96.44±1.00**†	**81.68±0.80**†	48.28±0.74†	**154.64±1.32**†	199.96±1.17	358.56±3.14†
	10	96.96±0.98†	81.92±0.81†	**48.24±0.52**†	154.84±1.28†	**199.24±1.27**†	358.52±3.84†
	100	96.92±1.26	81.80±0.65†	48.52±0.65†	155.64±1.35†	199.72±0.84	**358.36±2.96**†
	1000	97.12±1.54	81.96±0.61†	48.52±0.71	156.64±1.15†	200.36±1.44	361.68±2.04†
	10000	97.40±0.91	81.84±0.62†	48.84±0.62	158.08±1.29†	201.92±0.91	368.32±2.08†
Execution Time (s)	5	100.76±4.94	72.60±2.66	165.91±18.56	128.82±6.67	139.14±8.70	195.61±13.07
	10	79.42±4.61	49.43±3.90	148.77±15.55	110.18±9.89	120.47±7.46	171.84±11.16
	100	69.23±4.39	34.70±3.60	137.72±8.81	90.47±9.89	104.25±9.32	149.49±8.47
	1000	65.02±5.05	33.48±5.90	127.98±9.88	83.23±8.14	103.89±16.37	172.82±50.10
	10000	59.54±4.51	30.91±4.08	114.70±9.05	78.88±7.87	97.44±15.12	169.07±56.89
Sequence Length	5	3.13±3.13	3.01±3.01	2.97±0.56	3.33±0.55	3.55±0.69	3.05±0.69
	10	3.28±3.28	3.38±3.38	3.05±0.71	3.19±0.73	3.67±0.81	3.02±0.64
	100	3.97±3.97	3.69±3.69	3.27±0.67	3.65±0.67	3.95±0.63	2.87±0.47
	1000	4.08±4.08	3.78±3.78	3.53±0.76	3.74±0.60	4.07±0.56	2.88±0.57
	10000	4.19±4.19	3.87±3.87	3.70±0.70	3.79±0.55	4.09±0.46	2.83±0.68

†Statistically significant improvement over standard SSHH and Multi-SSHH with a $p < 0.05$ t-test, a two-sided significance level and 24 degrees of freedom

A method to reduce the ability for a single parallel SSHH model's HMMs to skew the success counts when the HMMs are combined is to only enable parallel SSHH models to update their local HMM success score when the new solution is better than the global best. Each parallel SSHH model retains its own best found solution and updates this if a sequence improves this but will only increment their transitional success scores if this is better than the global best. The results from this small adjustment are shown in Table 4 whereby in terms of non-service distance and bus fleet size a sustained improvement is achieved over the results in Table 3. With this change the results are now a small improvement over a standard sequential SSHH implementation and Multi-SSHH solution quality as shown in Table 2. Note that higher numbers of asynchronous iterations leads to both improved non-service distances and bus fleet sizes, approximately 1000 asynchronous iterations. Also note that sequence sizes are reduced which may be a reason behind the improved results as shorter sequences are expected to have a greater probability of improving the current solution.

4.4 Sharing the Best Solutions

The alternative model to assess is instead of sharing state transition success counts across parallel SSHH models is to share the best found solution. A range of diverse behaviours in terms of sequence generation will be present in each parallel SSHH model with the best found solution to execute heuristic sequences upon. As previously, a range of asynchronous iterations will be tested to ascertain the best number to use with the results shown in Table 5. The first observation that can be made in comparison to the results from sharing HMM information as shown in Table 4 is that sharing the best solution provides a marked improvement in results both for non-service distance and buses utilised. Furthermore, comparing these results to standard sequential SSHH and non-sharing parallel Multi-SSHH results in Table 2 a significant improvement can be observed, up to a 10% reduction in non-service distance and 4% reduction in bus fleet size. Effectively, enabling diversity in terms of parallel SSHH models with differing HMM behaviours with access to the best found solution increases the ability to exit a local optima. Indeed, some of the HMM may not generate particularly effective sequences of heuristics in terms of general optimisation but could be useful in some instances. However, note that in contrast to sharing HMM information, smaller degrees of asynchronous iterations provides the best results indicating that quicker access to a new globally best found solutions is important. Using a low-level of asynchronous iterations does increase the runtimes, as previously discussed, a higher degree of synchronisation causes more waiting time for parallel implementations of SSHH to reach the same iterations reducing CPU occupancy. Consequently, whilst using a low-level of asynchronous iterations gets the best results, a higher number could run for more iterations in the same time potentially getting better results. Also, note the shorter average sequence lengths in comparison to the results in Table 4 when sharing HMM information.

4.5 Comparison of Ensemble SSHH Sizes

It has been observed that a parallel SSHH model whereby the best solution found is sent to each parallel thread to use is most effective. The differing behaviours of the individual HMMs operate in the form of an ensemble approach. Due to the processor used being capable of 16 simultaneous threads the ensemble size was set to this number. A question remains as to if this is the best ensemble size to use or should there be a greater number of parallel SSHH models in the ensemble or fewer which could then run more iterations. Subsequently, to answer this question a range of ensemble sizes of parallel SSHH model will be tested for their effectiveness. The best solution sharing parallel SSHH model will be used with ten asynchronous iterations. Note that if the number of parallel models is increased, the iterations each runs for is similarly reduced ensuring only five million sequences are generated in total.

Table 6: Average optimised electric bus fleet non-service distance travelled, bus fleet sizes, runtimes and sequences lengths for range of permissible asynchronous iterations when sharing best found solutions for range of SSHH parallel ensembles using 10 asynchronous iterations between parallel sharing of best solutions

	Parallel SSHH Implementations	Electric Bus Fleet Routing Scenario					
		A	B	C	D	E	F
Non-Service Distance (Km)	8	**3009.18±64.25**†	2059.86±70.89†	1045.31±64.31†	5470.45±146.73†	4894.08±122.82†	10 757.27±491.74†
	16	3072.37±73.96†	2051.06±63.85†	**1040.87±74.65**†	5464.44±116.19†	4864.93±118.58†	10 731.81±358.38†
	32	3024.91±85.29†	2040.39±61.79†	1052.49±58.23†	**5460.12±86.42**†	**4859.98±134.80**†	**10 720.98±290.02**†
	64	3033.35±86.44†	**2028.25±76.55**†	1046.41±56.87†	5496.55±113.12†	4940.67±120.61	10 833.45±297.85†
	128	3042.01±70.91†	2047.58±56.06†	1024.35±44.33†	5494.85±157.45†	4926.86±110.00	11 080.59±238.26†
Bus Fleet Size	8	**96.16±1.11**†	81.80±0.76†	48.40±0.71†	154.52±1.42†	199.44±1.36†	359.72±5.50†
	16	96.96±0.98†	81.92±0.81†	**48.24±0.52**†	154.84±1.28†	**199.24±1.27**†	**358.52±3.84**†
	32	96.52±1.16†	81.80±0.76†	48.36±0.64†	**154.44±0.96**†	**199.24±1.48**†	358.80±3.25†
	64	96.36±0.81†	**81.36±0.81**†	48.52±0.87†	154.68±1.22†	199.92±1.00	359.64±3.28†
	128	96.44±0.92†	81.68±0.69†	48.44±0.65†	154.88±1.36†	199.80±1.26†	361.60±2.84†
Execution Time (s)	8	113.04±4.15	71.33±7.41	239.50±101.38	143.54±14.02	160.53±14.32	236.80±35.52
	16	79.42±4.61	49.43±3.90	148.77±15.55	110.18±9.89	120.47±7.46	171.84±11.16
	32	68.21±3.08	43.66±2.39	110.93±10.38	98.78±2.98	106.88±5.90	151.84±11.77
	64	62.34±2.66	40.53±1.27	96.38±7.86	92.33±3.98	101.16±3.42	155.81±12.10
	128	60.86±2.68	39.10±1.21	88.29±5.62	91.85±3.01	100.34±3.40	159.45±10.94
Sequence Length	8	3.39±3.39	3.35±3.35	3.04±0.85	3.32±0.87	3.47±1.10	2.96±0.75
	16	3.28±3.28	3.38±3.38	3.05±0.71	3.19±0.73	3.67±0.81	3.02±0.64
	32	3.35±3.35	3.00±3.00	2.93±0.42	3.17±0.45	3.73±0.32	3.13±0.66
	64	3.34±3.34	2.89±2.89	3.08±0.33	3.23±0.35	3.61±0.42	3.37±0.36
	128	3.28±3.28	2.87±2.87	2.96±0.25	3.23±0.16	3.60±0.29	3.22±0.34

†Statistically significant improvement over standard SSHH and Multi-SSHH with a $p < 0.05$ t-test, a two-sided significance level and 24 degrees of freedom

The results from these experiments are shown in Table 6 whereby it can be first observed that in terms of non-service distances and bus fleet sizes the results are all relatively similar. This indicates the number of parallel SSHH models is not highly important although note that higher numbers of parallel SSHH models use less iterations. Overall, with the larger problems a size of 16-32 parallel SSHH models provides the better results. However, in terms of the execution timings the larger numbers of parallel SSHH models use less time. This is due to greater occupancy of the CPU with more parallel threads executing and there being fewer synchronisation steps since less overall iterations are being used but the number of asynchronous steps remains the same at ten iterations.

5 CONCLUSIONS

In a world of ubiquitous multi-core parallel processors this paper has considered the sequential nature of the Sequence-Based Selection Hyper-Heuristic (SSHH) algorithm and parallel methods to improve its performance. It was also postulated that multiple SSHH models could improve upon the optimisation accuracy borrowing principles from the ensemble methods used to improve classifier performance. The process of how parallel methods share information between themselves is vitally important to performance hence two parallel SSHH models were considered, sharing of the HMM state transition probabilities and sharing of the best solution.

For a complex electric bus scheduling optimisation problem the parallel SSHH model whereby the best solution found is shared to all parallel SSHH models proved superior to sharing HMM information and sequential SSHH. A potential reason for this is that in some respects key relationship information is lost when combining the parallel HMMs leading to what could in effect be called an *average* HMM. A significant speedup in SSHH is also achieved due to the parallelism. In terms of how many parallel instantiations of SSHH should be run when sharing best found solutions, or the size of the ensemble, a size of 16 or 32 provides the best results. However, this is most likely dictated by the number of processor cores available. If there are more processor cores available then it would make sense to use them. Moreover, the work presented in this paper ensured that five million sequences were always generated by reducing the iterations of increased SSHH instantiations but this does not need to be the case if not seeking an algorithmic speedup.

Future work will consider further parallel models such as a multi-test approach to better assess heuristic sequence quality or a population of solutions method common to meta-heuristics. Additionally, an improved *broadcast* SSHH parallel model could remove the synchronisation steps which reduced CPU occupancy.

ACKNOWLEDGMENTS

Supported by Innovate UK [grant no. 10007532] and City Science.

REFERENCES

[1] Ekaterina Alekseeva, Mohand Mezmaz, Daniel Tuyttens, and Nouredine Melab. 2017. Parallel multi-core hyper-heuristic GRASP to solve permutation flow-shop problem. *Concurrency and Computation: Practice and Experience* 29, 9 (2017), e3835.

[2] Alex R Bertels and Daniel R Tauritz. 2016. Why asynchronous parallel evolution is the future of hyper-heuristics: A cdcl sat solver case study. In *Proceedings of the 2016 on genetic and evolutionary computation conference companion*. 1359–1365.

[3] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society* 64, 12 (2013), 1695–1724.

[4] Darren M Chitty. 2021. A partially asynchronous global parallel genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1771–1778.

[5] Darren M Chitty and Ed Keedwell. 2022. Defining a Quality Measure Within Crossover: An Electric Bus Scheduling Case Study. In *Artificial Evolution, EA 2022*. 97–110.

[6] Darren M Chitty, William B Yates, and Ed Keedwell. 2022. An edge quality aware crossover operator for application to the capacitated vehicle routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 419–422.

[7] Peter Cowling, Graham Kendall, and Eric Soubeiga. 2000. A hyperheuristic approach to scheduling a sales summit. In *International conference on the practice and theory of automated timetabling*. Springer, 176–190.

[8] Georges A Croes. 1958. A method for solving traveling-salesman problems. *Operations research* 6, 6 (1958), 791–812.

[9] Tansel Dokeroglu and Ahmet Cosar. 2016. A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem. *Engineering Applications of Artificial Intelligence* 52 (2016), 10–25.

[10] Marco Dorigo and Luca Maria Gambardella. 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation* 1, 1 (1997), 53–66.

[11] John H Drake, Ahmed Kheiri, Ender Özcan, and Edmund K Burke. 2020. Recent advances in selection hyper-heuristics. *European Journal of Operational Research* 285, 2 (2020), 405–428.

[12] Russell Eberhart and James Kennedy. 1995. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*. IEEE, 39–43.

[13] Mohammad Moein Fazeli, Yaghoub Farjami, and Mohsen Nickray. 2019. An ensemble optimisation approach to service composition in cloud manufacturing. *International Journal of Computer Integrated Manufacturing* 32, 1 (2019), 83–91.

[14] John J Grefenstette. 1981. Parallel adaptive algorithms for function optimization. *Vanderbilt University, Nashville, TN, Tech. Rep. CS-81-19* (1981).

[15] Lars Kai Hansen and Peter Salamon. 1990. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence* 12, 10 (1990), 993–1001.

[16] John H Holland. 1975. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press.

[17] Baldomero Imbernón, Antonio Llanes, José-Matías Cutillas-Lozano, and Domingo Giménez. 2020. HYPERDOCK: Improving virtual screening through parallel hyperheuristics. *The International Journal of High Performance Computing Applications* 34, 1 (2020), 30–41.

[18] Ahmed Kheiri and Ed Keedwell. 2017. A hidden markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems. *Evolutionary computation* 25, 3 (2017), 473–501.

[19] Coromoto León, Gara Miranda, and Carlos Segura. 2009. Hyperheuristics for a dynamic-mapped multi-objective island-based model. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living: 10th International Work-Conference on Artificial Neural Networks, IWANN 2009 Workshops, Salamanca, Spain, June 10-12, 2009. Proceedings, Part II 10*. Springer, 41–49.

[20] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. 2003. Iterated local search. In *Handbook of metaheuristics*. Springer, 320–353.

[21] Helena Ramalhinho Lourenço, Olivier C Martin, and Thomas Stützle. 2019. Iterated local search: Framework and applications. In *Handbook of metaheuristics*. Springer, 129–168.

[22] David Meignan. 2011. An evolutionary programming hyper-heuristic with co-evolution for CHeSC11. In *The 53rd Annual Conference of the UK Operational Research Society (OR53)*, Vol. 3.

[23] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. 2011. A new hyper-heuristic implementation in HyFlex: a study on generality. In *Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory & Application*. 374–393.

[24] Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J Parkes, Sanja Petrovic, et al. 2012. Hyflex: A benchmark framework for cross-domain heuristic search. In *Evolutionary Computation in Combinatorial Optimization: 12th European Conference, EvoCOP 2012, Málaga, Spain, April 11-13, 2012. Proceedings 12*. Springer, 136–147.

[25] Paola P Oteiza, Juan I Ardenghi, and Nélida B Brignole. 2021. Parallel hyper-heuristics for process engineering optimization. *Computers & Chemical Engineering* 153 (2021), 107440.

[26] Petr Pospichal, Jiri Jaros, and Josef Schwarz. 2010. Parallel genetic algorithm on the cuda architecture. In *Applications of Evolutionary Computation: EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*. Springer, 442–451.

[27] Prapa Rattadilok, Andy Gaw, and Raymond SK Kwan. 2005. Distributed choice function hyper-heuristics for timetabling and scheduling. In *Practice and Theory of Automated Timetabling V: 5th International Conference, PATAT 2004, Pittsburgh, PA, USA, August 18-20, 2004, Revised Selected Papers 5*. Springer, 51–67.

[28] Diego A Rodriguez, Paola P Oteiza, and Nélida B Brignole. 2019. An urban transportation problem solved by parallel programming with hyper-heuristics. *Engineering Optimization* 51, 11 (2019), 1965–1979.

[29] Nasser R Sabar and Graham Kendall. 2015. Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems. *Information Sciences* 314 (2015), 225–239.

[30] Carlos Segura, Eduardo Segredo, and Coromoto Leon. 2012. Analysing the adaptation level of parallel hyperheuristics applied to multiobjectivised benchmark problems. In *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 138–145.

[31] Reiko Tanese. 1987. Parallel genetic algorithm for a hypercube. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987.

[32] Reiko Tanese. 2013. Parallel genetic algorithm for a hypercube. In *Genetic Algorithms and Their Applications*. Psychology Press, 177–183.

[33] Willem Van Onsem and Bart Demoen. 2013. Parhyflex: A framework for parallel hyper-heuristics. In *Proceedings of the 25th Benelux Conference on Artificial Intelligence*, Vol. 28. 231–238.

[34] Pablo Vidal and Enrique Alba. 2010. Cellular genetic algorithm on graphic processing units. *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)* (2010), 223–232.