2015 International Conference on Computational Science and Computational Intelligence

Evolutionary Ensemble Strategies for Heuristic Scheduling

Thomas Philip Runarsson
School of Engineering and Natural Science
University of Iceland, Iceland
Email: tpr@hi.is

***Abstract*—An ensemble of single parent evolution strategies voting on the best way to construct solutions to a scheduling problem is presented. The ensemble technique applied is analogous to those described in the machine learning literature. A set of individuals vote on the best way to construct solutions and so collaborate with one another. An experimental study illustrates the superiority of the ensemble strategy over evolutionary strategies where individuals do not collaborate.**

***Keywords*-Evolutionary algorithms; ensembles; scheduling;**

I. Introduction

Scheduling is the task of allocating a set of jobs to a set of machines to meet a predefined criteria. Due to its importance in production management, scheduling has been widely researched since the industrial revolution. For some scheduling problems, such as the job-shop problem considered here, the computational complexity increases exponentially with the increasing number of jobs and machines. For this reason numerous heuristic methods for finding approximate solutions have been proposed. A popular approach is to apply handcrafted dispatching rule, construction heuristics, for a given scheduling task [1]. Dispatching heuristics essentially assign jobs based on attributes of the current partial schedule or attributes of the resulting partial schedule resulting from the job dispatched. For example, the job dispatched may be the one with the shortest processing time, most work remaining or a combination of different attributes (composite dispatching rules). Dispatching rules can therefore be considered to be a hypothesis, a classifier, for choosing the most appropriate job to dispatch, given the attributes of the current partially constructed schedule.

In machine learning ensemble methods use a finite set of alternative hypothesis resulting from the same base learning algorithm [2]. Perhaps one the best examples of this is the random forest [3], where an ensemble of decision trees are applied. The idea with the ensemble is that a set of weak hypothesis combined together can achieve a better classification accuracy. Multiple-classifiers are a similar technique where the hypothesis are produced using different machine learning algorithms. The general idea of using an ensemble of algorithms in optimization is that the performance of a single algorithm is sensitive to its various parameter settings. Using an ensemble of algorithms or even the same algorithm with different parameter settings would then compensate for this lack of performance. In the case of an ensemble classifiers each hypothesis will contribute towards the classification. Ensembles in evolutionary algorithms produce points in the search space which are either copied to the next generation or deleted based on a performance criteria [4]. This ensemble of algorithms essentially compete or complement each other but do not collaborate.

For the work presented here the term ensemble is used in the same context as in the machine learning literature. The same base learning algorithms, the evolution strategy $(1+1)$CMA-ES, is applied to generate composite dispatching rules for the construction of solutions to the job-shop scheduling problem. The base learners are λ evolution strategies running in parallel, each one itself a single composite dispatching rule (hypothesis) for constructing a schedule at each generation (iteration of the evolution strategy). The ensembles are created by taking a random sub-sample, without replacement, from the λ dispatching rules represented by the population of evolution strategies. This ensemble of dispatching rules is then used to build schedules. The aim is to obtain better performance using an ensemble which could not be achieved by a single dispatching rule. That is, an individual evolution strategy. The goal is to produce an ensemble evolution strategy which collaborates in order to make up for their individual weaknesses. The application under consideration essentially involved a classifier for constructing solutions. However, unlike ensemble machine learning algorithms, such as AdaBoost [5], there is no supervisor available to label the job dispatches for training. Training will be achieved by selective pressure alone, in the usual evolutionary manner.

The paper is organized as follows. First we formally define the job shop scheduling problem and heuristics used to construct solutions. This is followed by a brief description of the $(1+1)$CMA-ES as implemented in [6]. The proposed ensemble evolution strategy developed is described in section IV. This is followed by an experimental study of its performance on some 30 medium size scheduling problems. The paper concludes with a summary and main conclusions.

II. Scheduling Heuristics

The scheduling problem considered here is the job-shop problem. It consists of a set of jobs that need to be processed on a set of machines. The problem assumes that a machine

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.91

can only process one job at a time and that the processing of a job, referred to as an operation, cannot be interrupted. In an $n' \times m'$ job-shop problem, n' jobs must be processed on m' machines [1]. The jobs are scheduled as a chain of operations where each operation needs to be processed during a given time period on each machine. The objective is to find the set of operations (a schedule) that gives the shortest completion time.

Heuristics algorithms for scheduling are typically either a construction or improvement heuristics. The improvement heuristic starts with a complete schedule and then tries to find similar, but better schedules. A construction heuristic starts with an empty schedule and adds one job at a time until the schedule is complete. This is the approach taken here.

In order to apply a dispatching rule a number of attributes of the schedule being built must be computed. Figure 1 shows an example of such a partial schedule for a six job and six machine job-shop problem. The numbers in the boxes represent the job identification j. The width of the box illustrates the processing times for a given job for a particular machine M_i (on the vertical axis). The dashed boxes represent the resulting partial schedule for when a particular job is scheduled next. For example, if the job with the shortest processing time were to be scheduled next then job 4 would be dispatched. The placement of the job is such that it is placed in the earliest possible time in the current partial schedule. Attributes are used to grasp the essentials of the current state of the schedule. The attributes of particular interest were obtained from commonly used single priority-based dispatching rules [7]. Some attributes are directly observed from the partial schedule. The temporal scheduling attributes applied here for a job j to be dispatched on machine M_i are given in Table I. A dispatching rule may

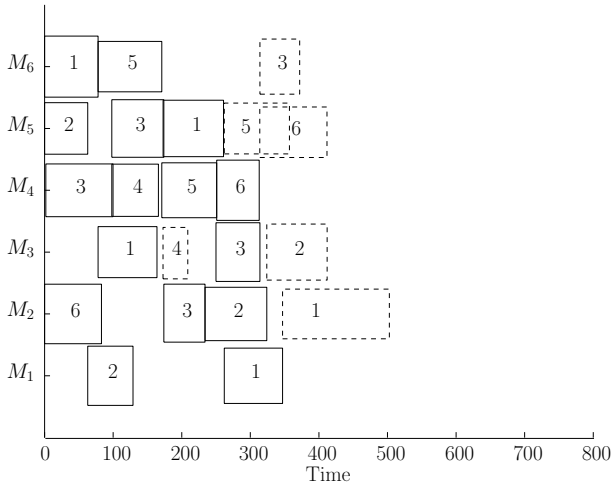

Figure 1. Gantt chart representing a partial schedule. The solid labelled boxes represent scheduled jobs and their job identification. The dashed boxes represent possible next step dispatches.

Table I
ATTRIBUTES FOR THE JSP WHERE JOB j ON MACHINE M_i GIVEN THE RESULTING TEMPORAL SCHEDULE AFTER DISPATCHING.

$\boldsymbol{\phi}(j)$	Feature description
$\phi_1(j)$	processing time
$\phi_2(j)$	start-time
$\phi_3(j)$	end-time
$\phi_4(j)$	when machine is next free
$\phi_5(j)$	current makespan
$\phi_6(j)$	work remaining
$\phi_7(j)$	most work remaining
$\phi_8(j)$	slack time for machine
$\phi_9(j)$	slack time for all machines
$\phi_{10}(j)$	slack weighted w.r.t. number of tasks assigned
$\phi_{11}(j)$	time job had to wait
$\phi_{12}(j)$	size of slot created by assignment
$\phi_{13}(j)$	total processing time for job

need to perform a one-step look-ahead and observe attributes of the partial schedule in order to make a decision, for example by observing the current makespan for the partially completed schedule. Other dispatching rules use attributes directly from the current partial schedule, for example by assigning jobs with most total processing time remaining. A composite dispatching rule combines the attributes of the post-decision schedule to create an evaluation function for each job j dispatched. That is,

$$eval(j) = \sum_{i=1}^{13} y_i \phi_i(j) \tag{1}$$

and the job whose evaluation value is highest is the one dispatched (breaking ties randomly). We will now describe an evolution strategy used to search for an effective set of weights $\mathbf{y}$.

III. $(1+1)$CMA-ES

The $(1+1)$ Covariance Matrix Adaptation Evolutionary Strategy, $(1+1)$CMA-ES, is a single *parent* search strategy. A single iterations of this search strategy will now be described, but the reader is referred to [6] for a more complete description. The parent $\mathbf{x}$ is replicated (imperfectly) to produce an offspring $\mathbf{y} = \mathbf{x} + \sigma\mathcal{N}(\mathbf{0}, \boldsymbol{C})$, where σ is a global step size and $\boldsymbol{C}$ the covariance matrix of the zero mean Gaussian distribution. The replication is implemented as follows:

$$\mathbf{z} = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{2}$$

$$\mathbf{s} = \boldsymbol{A}\mathbf{z} \tag{3}$$

$$\mathbf{y} = \mathbf{x} + \sigma\mathbf{s} \tag{4}$$

where the covariance matrix has been decomposed into Cholesky factors $\boldsymbol{A}\boldsymbol{A}^{\intercal}$. The normally distributed random vector $\mathbf{z}$ is sampled from the standard normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The success probability of this replication is updated by

$$\bar{p}_{succ} \leftarrow (1 - c_p)\bar{p}_{succ} + c_p \mathbb{1}\big[f(\mathbf{y}) \leq f(\mathbf{x})\big] \tag{5}$$

where $f(\cdot)$ if the objective (fitness) function which will be minimized. Here $\mathbb{1}[\cdot]$ is the indicator function and takes the value one if its argument is true otherwise zero. The parameter c_p is the learning rate ($0 < c_p \leq 1$) and is set to $1/12$. The initial value for $\bar{p}_{succ} = 2/11$ which is also the target success probability p^t_{succ}. Following the evaluation of the success probability the global step size is updated by

$$\sigma \leftarrow \sigma \exp\left(\frac{\bar{p}_{succ} - p^t_{succ}}{d(1 - p^t_{succ})}\right) \tag{6}$$

where $d = 1 + n/2$ and n the number of objective variables. The initial global step size will be problem dependent but should cover the intended search space. All of these default parameter setting are discussed in [6].

If the replication was successful, that is $f(\mathbf{y}) \leq f(\mathbf{x})$, then $\mathbf{y}$ will replace the parent search point $\mathbf{x}$. Furthermore, the Cholesky factors $\boldsymbol{A}$ will be updated. Initially $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ are set to the identity matrix and $\mathbf{s}$ set to $\mathbf{0}$. The update is then as follows [6]:

1. If $\bar{p}_{succ} < p^t_{succ}$ then $\mathbf{s} \leftarrow (1-c)\mathbf{s} + \sqrt{c(2-c)}\boldsymbol{A}\mathbf{z}$ and set $\alpha \leftarrow (1 - c_{cov})$
   else $\mathbf{s} \leftarrow (1-c)\mathbf{s}$ and set $\alpha \leftarrow 1 - c^2_{cov}c(2-c)$.
2. Compute $\mathbf{w} \leftarrow \boldsymbol{A}^{-1}\mathbf{s}$ and set $a = \sqrt{1 + c_{cov}\|\mathbf{w}\|^2/\alpha}$
3. $\boldsymbol{A} \leftarrow \sqrt{\alpha}\boldsymbol{A} + \sqrt{\alpha}(a-1)\mathbf{s}\mathbf{w}^\intercal/\|\mathbf{w}\|^2$
4. $\boldsymbol{A}^{-1} \leftarrow \frac{1}{\sqrt{\alpha}}\boldsymbol{A}^{-1} - \frac{1}{\sqrt{\alpha}\|\mathbf{w}\|^2}\big(1 - 1/a\big)\mathbf{w}\big[\mathbf{w}^\intercal\boldsymbol{A}^{-1}\big]$.

The default setting for the covariance weight factor $c_{cov} = 2/(n^2+6)$ and $c = 2/(2+n)$. $\boldsymbol{A}^{-1}$ requires $\Theta(n^2)$ time, whereas a factorization of the covariance matrix requires $\Theta(n^3)$ time. The Cholesky version of the $(1+1)$CMA-ES is therefore computationally more efficient.

A set of λ such $(1+1)$CMA-ES will now be used to construct an ensemble evolution strategy.

IV. Ensemble Evolutionary Search

Consider a population of covariance matrix adaptation evolutionary strategy algorithms described in the previous section. Each individual strategy will adapt its search distribution, its search strategy, independently of one another. These strategies will also compete and successful $(1+1)$CMA-ESs overwrite unsuccessful ones. This approach to search may in itself be considered an ensemble strategy since each $(1+1)$CMA-ES will be using its own search distribution. One could then also argue that evolutionary algorithms should be considered to be an ensemble search strategy. The population is after all an ensemble. Furthermore, one may consider recombination as a form of collaboration and boosting. This is, however, not the approach pursued here.

The ensemble created at each generation will be a subset $\mathcal{S}$ of the point vectors $\mathbf{y}^1, \ldots, \mathbf{y}^\lambda$ proposed by each of the λ $(1+1)$CMA-ES in parallel. A given vector k within $\mathcal{S}$ will vote for the job to be dispatched by performing a one-step lookahead for a given job dispatch. That is, a classifier C_k

$$C_k = \operatorname*{argmax}_j \sum_{i=1}^{13} y^k_i \phi_i(j), \quad k \in \mathcal{S} \subset \{1, \ldots, \lambda\} \tag{7}$$

that returns the job to dispatch by the k-th composite dispatching rule. However, the actual job dispatched will be the one proposed by a *majority vote*. Majority vote is perhaps the simplest to implement and assumes no prior knowledge of the composite dispatching rules used. The majority vote is the one with the most predicted job label j,

$$C^* = \operatorname*{argmax}_j \sum_{k:C_k=j} 1 \tag{8}$$

where ties are broken randomly.

One must now consider how to generate the subset $\mathcal{S}$ and decide on the fitness for the k-th composite dispatching rule $f(\mathbf{y}^k)$. Its fitness will clearly depend on the quality of the schedule constructed by the ensemble. The size of the ensemble $m = |\mathcal{S}|$ must also be considered and how the set is sampled. Since no prior knowledge of what type of ensemble is most likely to succeed, the ensemble will be sampled randomly from the population without replacement. The performance of any given $(1+1)$CMA-ES will now critically depend on the ensemble. In order to minimize the noise created by such an evaluation, each $(1+1)$CMA-ES will participating in m such independently drawn ensembles. The number of schedules built will then be in total λ. Let us denote the ensembles having the k-th $(1+1)$CMA-ES by the sets of ensembles $\mathcal{S}_e$, $e = 1, \ldots, m$. Furthermore, let the quality of the schedule build by the ensemble be denoted by f_e (the resulting makespan). Then, let the fitness of point vector $\mathbf{y}^k$ be the best schedule created by the set of ensembles it participated within, that is

$$f(\mathbf{y}^k) = \min_{e:k\in\mathcal{S}_e} f_e(\mathbf{y}^k) \tag{9}$$

This is then also the fitness used to update the success rate $\bar{p}_{succ}$ and when successful will replace the parent point $\mathbf{x}$. Once each $(1+1)$CMA-ES has been evaluated a simple truncation selection is performed on all the strategies. This is where the worse half of the population of $(1+1)$CMA-ES is deleted and the better half is doubled.

V. Experimental Study

The goal of this experimental study is to demonstrate the performance boost achieved when individual composite dispatching rules work together in an ensemble. More importantly how the individual benefits when collaborating with other individuals within the population. The collaborative population of individuals will work together in building solutions to scheduling problems. Each individual is basically a linear classifier and votes on the particular job to dispatch. A set on m individuals are taken from the population of λ individuals, here we will use $\lambda = 100$ and $m = 5$, and each will collaborate with different individuals each time exactly m times. This ensemble is sampled from the population, each time without replacement. The ensembles build solution using a majority vote and the performance

of a single individual will be based on its best ensemble performance.

As an additional comparison individual ensembles will be evolved. That is, each $(1+1)$CMA-ES will evolve an ensemble of m composite dispatching rules. The number of variables will, therefore, be $13 \times 5 = 65$. Given that ensembles boost the performance of the individuals, one would expect this approach to work equally so.

A. Experimental Setup

The test problems used in the study were created using the methodology proposed by [8]. The goal is to minimize the makespan, $C_{\max}$. The processing time of the jobs, on all the machines and all problems, is an integer uniformly distributed between 1 and 100. The processing order of every job is a random permutation. Every job has to visit every machine once. For our experiments we will generate 30 independent 10×10 problems. The problems are solved to optimality using a branch and bound algorithm developed by [9]. The optimum makespan is denoted $C^{\text{opt}}_{\max}$. Since the optimal makespan varies between problem instances the performance measure is the following,

$$\rho = \frac{C_{\max} - C^{opt}_{\max}}{C^{\text{opt}}_{\max}} \tag{10}$$

which indicates the relative deviation from optimality.

The three evolutionary algorithms compared are as follows.

1) Composite Dispatching Heuristic (CDH): The CDH is the direct parallel search of 100 $(1+1)$CMA-ES for a single composite dispatching rule. That is, 100 independent $(1+1)$CMA-ES runs are made. There is no communication between the different $(1+1)$CMA-ES of any form. The quality of an individual is based solely on its performance in building a schedule.

2) Ensemble Evolutionary Search (EES: The EES is also a direct parallel search of 100 $(1+1)$CMA-ES for a single composite dispatching rule. However, the quality of the individual is based on its best ensemble performance. An ensemble of 5 individuals will vote on how best to build a schedule. The best out of 5 such ensembles will define the individual's performance.

3) Ensemble of Composite Dispatching Heuristic (ECDH): The ECDH is direct parallel search of 100 $(1+1)$CMA-ES for an ensemble of 5 composite dispatching rule. There is no communication between the different $(1+1)$CMA-ES. The quality of an individual is based solely on the performance of the ensemble within a single individual evolution strategy. This ensemble uses also majority voting when dispatching jobs.

B. Comparison and results

Each of the algorithms is run 30 times on the 30 independently generated test problems. All search methods

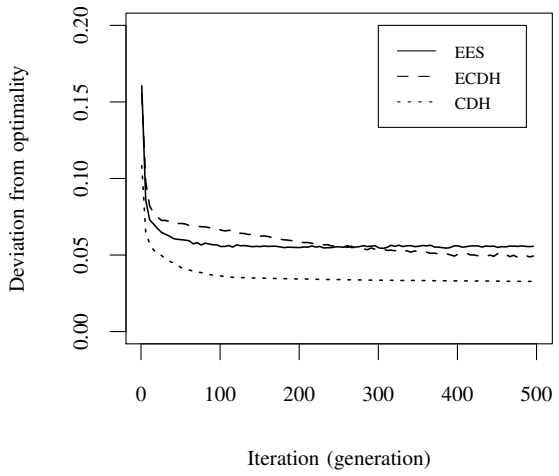

Figure 2. Average deviance from optimality as a function of generations for the three evolutionary algorithms compared.

are terminated after 500 generations. The average deviance from optimality of the best offspring $\mathbf{y}$ for all problems and runs is depicted in figure 2. From the figure one can see that the CDH algorithm produces the best individuals on average and its convergence is the fastest. A slightly slower convergence speed is observed for the EES with on average worse individuals. The reason for this is that the EES evaluation is noisy. An individual in the EES may be lucky and participate with successful ensembles in one generation and not so in the next. Furthermore, ties in the majority voting scheme will be broken randomly. The statistics reflect this and does not mean that the best schedule found for an EES run is on average worse than the CDH. On the contrary it will be better. The average performance of the best ECDH also experiences the same noise as the EES. However, its convergence is slower than the EES and will in the end surpass the EES. The slow convergence is due to the fact that its uses 5 times the number of parameters, i.e. 5×13. It surpassed the EES in the end as it uses consistently the same ensemble, unlike the EES which uses a random sample from the population. The ECDH is, therefore, less noisy than the EES.

It is interesting to investigate the average performance for particular problem instances. For sake of brevity only the first four out of thirty will be illustrated. Box plots for the deviation from optimality is shown in figure 3. There are in total 3×4 boxplots illustrated. The first three, from the left, are for the first problem instance for algorithms CDH, ECDH and EES respectively. Problems two to four follow in the same order. This figure illustrates first of all what is well known in the scheduling literature about dispatching rules, they can be successful but also fail terribly. In general a sin-

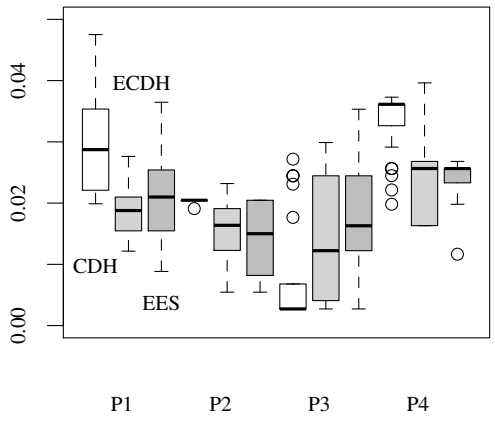

Figure 3. Comparison of the three different methods on the first four problem instances. Result are shown as box plots for the deviation from optimality.

gle composite dispatching rule (CDH) performs worse than the ensemble strategies. However, there will be exceptions, like problem P3. The performance distribution for EES and ECDH also appears different.

In order to determine the difference in performance over the 30 different problems, based on the 30 independent runs, a Wilcoxon rank sum test is used to see if any two performance distributions have equal medians. The number of times the three different algorithms are statistically different, and have a better median, is given in table II. When comparing the two ensemble strategies with the non-ensemble strategy the results for more than half the problems are statistically different and the majority are in favour of the ensemble methods. When comparing the two ensemble methods only 5 out of 30 are statistically different and all are in favour of ECDH.

VI. Summary and Discussion

The preliminary results presented here indicate that a significant performance boost may be achieved using an ensemble of dispatching rules to construct solution to scheduling problems. The approach taken is quite different to what has been proposed previously with ensembles in evolutionary computation [10]. The scheme is collaborative and in some sense co-evolutionary as the fitness of any given algorithm depends on the population of evolutionary algorithms. The ensemble evolution strategy draws on principles of ensembles in ensemble machine learning [4].

The direct evolution of the ensemble set (ECDH) has a slight advantage over the collaborative or co-evolutionary ensemble EES algorithm. This indicates that a still better design of the EES should be possible. This will require a better understanding of how the individuals co-evolve and is currently under investigation. Furthermore, how this approach may be extended to other problem domains in general is of particular interest.

Table II
The times a method has a better median performance when the medians are statistically different for the 30 problems.

	EES	ECDH	CHD
EES	-	0/5	11/16
ECDH	5/5	-	18/23
CDH	3/16	3/23	-

References

[1] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Prentice Hall, 2008.

[2] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research*, pp. 169–198, 1999.

[3] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[4] R. Mallipeddi, S. Mallipeddi, and P. N. Suganthan, "Ensemble strategies with adaptive evolutionary programming," *Information Sciences*, vol. 180, no. 9, pp. 1571–1581, 2010.

[5] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[6] T. Suttorp, N. Hansen, and C. Igel, "Efficient covariance matrix update for variable metric evolution strategies," *Machine Learning*, vol. 75, no. 2, pp. 167–197, 2009.

[7] S. S. Panwalkar and W. Iskander, "A survey of scheduling rules," *Operations Research*, vol. 25, no. 1, pp. 45–61, 1977.

[8] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[9] P. Brucker, *Scheduling algorithms*, 5th ed. Springer, 2007.

[10] G. Karafotias, M. Hoogendoorn, and A. Eiben, "Parameter control in evolutionary algorithms: Trends and challenges," *Evolutionary Computation, IEEE Transactions on*, vol. 19, no. 2, pp. 167–187, April 2015.