# Combining Simplex with Niche-based Evolutionary Computation for Job-Shop Scheduling

Syhlin Kuah and Joc Cing Tay*
Evolutionary and Complex Systems Programme, Nanyang Technological University
Blk N4, #2a-32 Nanyang Avenue Singapore 639798
*Email: asjctay@ntu.edu.sg

## ABSTRACT
We propose a hybrid algorithm (called ALPINE) between Genetic Algorithm and Dantzig's Simplex method to approximate optimal solutions for the Flexible Job-Shop Problem. Locally, Simplex is extended for the JSP linear program to reduce the number of infeasible solutions while solution quality is improved with an operation order search. Globally, a niche-based evolutionary strategy is employed to gain parallelization while solution diversity is maintained in two ways; composite dispatching rule-based population initialization and memory-based machine assignment. Performance results on benchmark problems show that ALPINE outperforms existing hybrid techniques with a new global optima found for the 10x7 Flexible Job Shop Problem.

## Categories and Subject Descriptors
I.2.8 [**Problem Solving, Control Methods and Search**]: Scheduling

## General Terms: Algorithms, Design, Performance.

## Keywords: Flexible Job-Shop Scheduling, Linear Programming, Niche-based Strategy, Hybrid Evolutionary Computation

## 1. INTRODUCTION
In modern manufacturing environments, proper production scheduling and planning is imperative to maximize efficiency given the limited resources that are available. The classical Job-Shop scheduling Problem (JSP) is a well-known machine scheduling model which features the essential objectives of manufacturing. As such there is abundant research on the JSP, particulary from the field of Operations Research [4]. The JSP may be described as follows: given *n* jobs, each consists of several operations that must be processed on *m* machines. Each operation uses one of *m* machines for a fixed duration. Each machine can process at most one operation at a time to completion without interruption. The operations of a given job have to be processed in a predetermined order, where no two or more operations from the same job can be processed simultaneously. The Flexible Job-Shop Problem (FJSP) extends this problem definition by removing the constraint of an one-to-one mapping of operations to machines and allowing an operation to be processed on more than one machine. The problem consists of finding a schedule of the operations on the machines, which yields the minimum makespan; the minimal completion time of all of the last job. Compared to the JSP, the decisions on machine assignment and operation precedence makes the FJSP strongly NP-hard [13]. In this paper, we propose a hybrid algorithm between Genetic Algorithm and a classical optimization method - Dantzig's Simplex method, to approximate optimal solutions to the FJSP.

The JSP can be modeled as a linear program to be solved by traditional Hopfield network applications [3]. However, Jain and Meeran [6] pointed out some setbacks of the Hopfield approach. Firstly, the network produces many infeasible solutions or schedules which violates the JSP constraints. Secondly, it easily gets trapped in local optima; hence optimal solutions even for small problems cannot be guaranteed. Dantzig (1947) designed the Simplex method to effectively solve Linear Programming problems (LP). We tackle the first setback by implementing an *extension* of Simplex for the same JSP linear program model to reduce the number of infeasible solutions, while the quality of solution is then further improved with an additional *operation-order search*. The combination of the extended Simplex and the operation-order search will serve as our local optimization mechanism. The second setback shows that using local optimization alone will produce inferior solutions in the target neighborhood. A common solution to this is the hybridization of a local search technique with a global search which is capable of spanning the search across different neighborhoods. As such, we apply the Genetic Algorithm (GA) (by Holland, Bremermann and Fraser) as it has been shown [9] to produce good results due to its ability to explore the search space of candidate solutions in parallel. However using a GA alone typically gives slow convergence to the best possible solution as it lacks local information to determine the most promising search direction. Hence, in this paper, we propose a GA-guided global search technique to speed up the rate of convergence while having the ability to escape from the local optimum through an extended Simplex with operation-order search. We are also aware that although a hybrid algorithm results in faster convergence, it is possible that this can interfere with the search for a global optimum by leading it to a premature convergence too quickly [10][16]. This will reduce the quality of solutions found. We address this issue with a niche-based GA architecture for maintaining candidate solution diversity.

This paper is organized as follows. Section 2 gives the formulation of the FJSP. Previous works on hybrid GA-search are reviewed in Section 3. Section 4 gives the design and motivation for the niche-based GA architecture. Section 5 describes the implementation details of our Hybrid model. Section 6 describes the technique to solve integer programs and its integration with our niche-based GA. Section 7 presents and analyzes experimental results. Finally we make some conclusions and discuss future work in Section 8.

## 2. PROBLEM FORMULATION

The FJSP with recirculation is formulated [13] as follows:

- There are $n$ jobs $J$=1, 2,..., $n$, indexed $i$.
- There are $m$ machines $M$=1, 2,... $m$, indexed $k$.
- Each operation, indexed $j$, can be processed on a set of machines $M_{ij}$. Therefore, $O_{ijk}$ denotes that operation $j$ of job $i$ is to be processed on machine $k$.
- $T_{ijk}$ defines the processing time of operation $j$ of job $i$ on machine $k$.
- Recirculation occurs when a job can visit a machine more than once.
- The objective is to find a schedule that has minimum makespan, that is, to minimize the completion time of the last job.

The following assumptions are made.

- All machines are available at time 0.
- All jobs are released at time 0.
- The order of operations in each job is predefined and cannot be modified.

The conditions above define a deterministic scheduling problem; where all parameters are known at the time of scheduling. A solution to a FJSP problem is a schedule where all of the above conditions are met. The decision variables of a solution is the start time of every operations, where:

- The start time of any operation of job $i$, processed on machine $k$, is defined as $S_{ik}$.

The constraints which relate the above variables will be discussed in detail under Section 5.1. The FJSP can be further divided into two subsets, namely Total FJSP (T-FJSP) and Partial FJSP (P-FJSP) [5]. T-FJSP allows each operation to be processed on any machine. In P-FJSP, only a proper subset of machines $M$ is allowed to process at least one operation $O_{ijk}$. We are able to solve the two kinds of FJSP using our proposed algorithm.

## 3. REVIEW OF HYBRID DESIGNS

Hybrid GAs have been applied extensively to yield better results [10][16][4] because of good complementary effects; that is, improving convergence in a promising area while maintaining a broad search for a global optimal in others. Nevertheless it has been found that too much local optimization can result in premature convergence [10]. This section will review different hybrid architectures and discuss the choice of an appropriate hybrid structure most appropriate for integrating Dantzig's Simplex and Holland's GA.

Four categories of hybrid architectures have been reviewed by Yen *et al.* [16]; namely, those that use Additional Operators, Asynchronous, Hierarchical, and Pipelining. Our choice of architecture depends on two factors; (a) the facility of integrating the Simplex with GA, and (b) the discrete nature of the FJSP.

The Additional Operators approach utilizes local search as an additional genetic reproduction operator. This operator works alongside with the conventional GA operator to generate the next population. This architecture is generally favored by researchers employing the probabilistic [16] or Nelder-Mead's Simplex [2][12] method into their hybrid model as the computational overhead introduced is very low. However, this approach is more appropriate for continuous functions, where a single Simplex search can be done on two or more individuals in the population. The same cannot be easily applied to a discrete optimization for the FJSP.

The Asynchronous Hybrid approach allows a GA and another optimization method to work on the same shared population separately. In this hybrid model, two or more search processes work independently on the same problem, updating the population if improvement is made by either one. Before we evaluate the feasibility of this second approach, allow us to briefly review the respective roles of GA and Simplex in solving the FJSP as separate entities. To solve the FJSP, GA requires at least two strings for its chromosome representation, namely machine assignment and operation order string [13]. The former will determine which machine an operation is processed on, while the latter determines the precedence of the operations. These two strings comprise a chromosome that specifies a job-shop schedule. On the other hand, only the machine assignment string is required to formulate a linear program, since the role of Simplex is simply to obtain the schedule with best minimized makespan without violating any precedence constraints. This therefore removes the need for an operation order string. In the context of Asynchronous Hybrids, we see that GA alone requires both sets while the Simplex only requires the machine assignment string. Hence while this architecture is possible for our Hybrid model, we do not consider this to be the best of choice.

Hierarchical Hybrids allows a GA and another optimization technique to work on two separate levels. Unlike Asynchronous Hybrids where both techniques work independently, here the optimization technique on one level will base its search on the one determined by GA on the previous level. This approach has some similarities with the Pipeline model discussed below.

The Pipelining Hybrid approach applies local and global search sequentially, where one creates data for the other [16]. We choose this approach for our hybrid model as the optimal result produced by Simplex will depend highly on the data, or machine assignments that it is given. GA will serve as a global search for the data, while the Simplex optimizes the given data efficiently. Unlike the Asynchronous Hybrid, the two techniques work hand in hand where the performance of local optimization will depend on the search by GA. Furthermore, only one string will be needed for our chromosome representation since they do not work independently.

## 4. A PIPELINE HYBRID MODEL

We will explain the roles of the GA and Simplex in a Pipeline Hybrid model for a single individual (chromosome) in a population. The *traditional* pipeline approach is to apply Simplex to every individual (generated by the GA) after each generation to obtain the minimal makespan (see Figure 1). In this model, Simplex is applied to improve the local convergence of individuals in the population, followed by genetic recombination to remove the local optima.

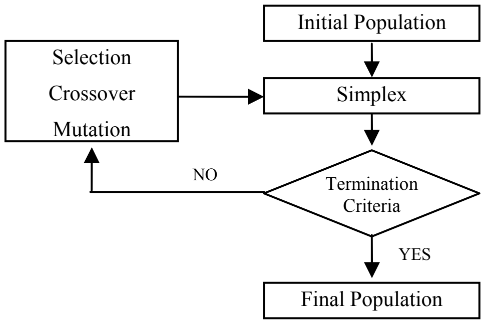

**Figure 1. Traditional Model of Simplex-GA Pipelining**

While this approach is straightforward, Mathias *et al*. [10] points out that using local search to process every individual can be computationally expensive. In our work, we tackle the issue of computational cost using a niche-based GA architecture to avoid processing every individual. This is discussed in Section 5.3.

## 5. SOLVING THE FJSP

This section will describe the implementation of our proposed Pipeline Hybrid model. We present an overview of the algorithm, followed by a discussion of features in our hybrid model; namely CDR-based Population Generator, Niche-based strategy and the Influenced Genetic Operator.

### 5.1 The ALPINE Algorithm

The ALPINE algorithm or *Applied Linear Programs In Niche-based Evolutionary computation*, is illustrated in Figure 2, and described as follows.

1. Initialization of population. A portion of the population is encoded with machine assignments produced by the CDR-based Population Generator [5]. The remainder is generated randomly as discussed under Section 5.2.

2. Evaluate fitness for *all* individuals. Fitness evaluation is done by PB-ILPIA with $O^2S$. Initialize the *makespan criteria* to be the best makespan obtained in the initial population. This makespan criteria is updated when better makespans are obtained in subsequent generations.

3. Sort the population according to their total processing time given in (7).

$$\sum_{i=1}^{n}\sum_{j=1}^{l}T_{ijk} \tag{7}$$

This is to facilitate clustering during subsequence generation of niches. Niche generation is executed by a *K*-means algorithm to split the individuals into *N* number of niches. Similarity is based on the individual's processing time for each operation. Similarity measure is the inverse of Euclidean distance between the selected individual and the mean of the niche. The Euclidean distance of an *n* –length encoding *X* and the mean *U* of the niche is given below.

$$\|X-U\|=\sqrt{\frac{1}{n}\sum_{k=1}^{n}\left(x_k-u_k\right)^2}$$

where $x_k$ and $u_k$ is the processing and mean processing times respectively of the assignment at index *k* of the chromosome representation.

4. For every niche, find the individual with a precomputed makespan; that is, an individual with a makespan value from previous generations that is unaltered by genetic operators. If none are found, randomly pick one and compute its fitness. If the makespan of the selected member (niche representative) has the same or better makespan than the makespan cateria, then evaluate the fitness of all members within the same niche. This allows a diversified search on one or more niches for a better makespan.

5. Rank all individuals based on the makespan, if available. If not, non-evaluated individuals, or individuals with unknown fitness, are sorted separately in the ranks of their estimated makespan. When the *belief space interval* is reached, updating of the operation belief space is carried out.

6. If *stagnant count* is reached, increase the makespan criteria by one unit. The stagnant counter tracks the number of iterations where no better solution has been obtained. By increasing the makespan criteria for the next generation, the algorithm is allowed to backtrack by evaluating less fitter individuals.

7. Apply recombination operators to generate the next population. This includes the use of the Influenced Genetic Operator which will be discussed under Section 5.4. Repeat steps 3 to 8 for the predefined number of generations.

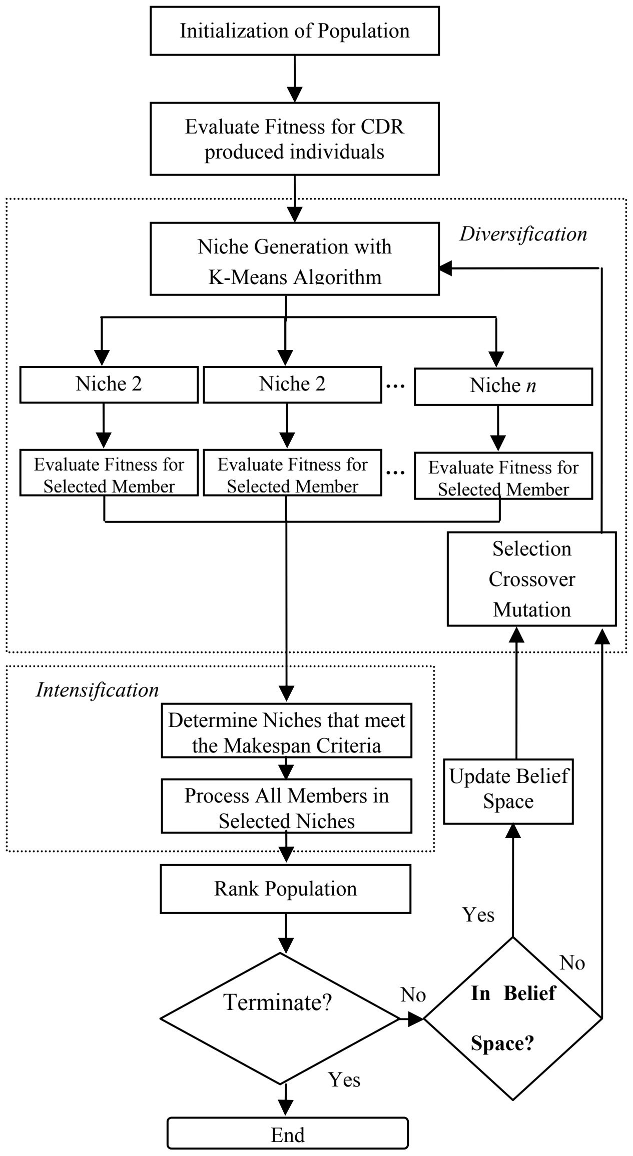
Initialization of Population
Evaluate Fitness for CDR produced individuals
Diversification
Niche Generation with K-Means Algorithm
Niche 2
Niche 2
Niche n
Evaluate Fitness for Selected Member
Evaluate Fitness for Selected Member
Evaluate Fitness for Selected Member
Selection Crossover Mutation
Intensification
Determine Niches that meet the Makespan Criteria
Process All Members in Selected Niches
Update Belief Space
Rank Population
Yes
No
Terminate?
No
In Belief Space?
Yes
End

**Figure 2. The ALPINE Algorithm**

## 5.2 Initial Population: Bootstrapping with CDR

The composite dispatching rules (CDR) bootstrap algorithm by Ho and Tay [5] is an efficient and effective way of producing good schedules. Results show that CDR alone produces better schedules for 3 out of 4 cases than those found by Kacem *et al.* [8]. The schedules produced by the bootstrap CDR are based on the following three rules: First-In-First-Out (FIFO), Shortest Processing Time (SPT), and Longest Processing Time (LPT). An additional rule - Random Dispatch, is added in our algorithm to further improve the diversity of a favorable encoding. This rule randomly calls any of the three former rules above. In our hybrid model, a portion of the machine assignments are produced by CDR-based Population Generator [5], while the rest are randomly assigned to maintain diversity desired for a global search.

## 5.3 Creating and Selecting Niches

Wei *et al.* [14] proposed a Niche Hybrid Genetic Algorithm (NHGA) based on the objectives of exploration and exploitation. NHGA boosts GA's diversification capabilities, as well as improve it's exploitation performance by hybridizing with Nelder-Mead's Simplex method. We adopt a similar approach to evaluate only those individuals that are grouped together in the same niche.

Our design is also inspired by the Continuous Hybrid Algorithm (CHA) by Chelouah *et al.* [2]; which emphasizes on solution *diversification* and *intensification*. In our algorithm, solution diversification is achieved through genetic operators to produce offsprings that are uniformly dispersed throughout the whole search space. To ensure similar offsprings are not created again, we use a memory enhancement technique; the Operation Belief Space [5], to influence our mutation. This prevents the search from being trapped in local optima. Separation of the population into niches helps us to group similar individuals together. We adopt the *K*-means algorithm to assign the individuals into their respective niches. As there will be one or more of such niches, selection of niches must be done through a criterion for the representative individual. If the representative fitness is equal or better than the best fitness currently found, the respective niche it belongs to will be evaluated for all individuals contained within it.

## 5.4 Diversity Maintenance with Influenced Genetic Operator

GENACE [5] implemented a belief space to store domain knowledge for pruning redundant information in the search space to guide it towards a global optimum. However, Branke [1] notes the high risk of misguiding evolution and preventing the diverse exploration of search space, so some form of diversity maintenance should be used in conjunction with memory. As such, we view the operation belief space (OB-Space) [5] as an alternative means to improve chromosomal diversity. The OB-Space is updated every *k* generations, where *n* elite individuals are taken into consideration. While previously in [5], the purpose of OB-Space was to create fitter individuals through mutation, we apply this feature to influence machine assignments for a particular operation that did not contribute to the elite individuals in previous generations. This is to ensure that diversity will always be maintained, and the algorithm does not easily get trapped in local optima. Next, we will present the design of the local optimization method and the integration with its GA counterpart.

## 6. LOCAL OPTIMIZATION

Design of the local optimization (for just ordering of operations on each machine - therefore a JSP) must consider the feasibility of integration with the GA counterpart. We divide this discussion into different sections. First we review the JSP formulation in a LP model. Second, we show how an additional search on the candidate solution of the LP can improve its quality. Third, we present our chromosome representation and how it is used to construct the FJSP-LP model.

## 6.1 A Linear Program for the FJSP

We begin by presenting a LP model for the JSP [3], which will be extended solve the FJSP. The model is as follows:

Minimize: $\sum_{i=1}^{n} S_{i,k_i}$

Subject to:

$S_{ik} - S_{ih} \geq T_{i,j-1,k}$ if operation$(i,j\text{-}1,h)$ precedes $(i,j,k)$ (1)
$S_{ik} \geq 0 \quad 1 \leq i \leq n$ (2)
$S_{pk} - S_{ik} \geq T_{i,j,k} \quad (O_{ijk}$ precedes $O_{pqk})$ (3)
$S_{ik} - S_{pk} \geq T_{p,q,k} \quad (O_{pqk}$ precede $O_{ijk})$ (4)
$S_{pk} - S_{ik} + H\times(1 - y_{ipk}) \geq T_{i,j,k}$ (5)
$S_{ik} - S_{pk} + H\times(y_{ipk}) \geq T_{p,q,k} \quad 1 \leq i,p \leq n\ ,\ 1 \leq k \leq m$ (6)

where $k_i$ is the machine which the last operation of job $i$ is assigned to, $S_{ik}$ is the start time of an operation of job $i$ on machine $k$, $T_{i,j,k}$ is processing time of operation $j$ of job $i$ on machine $k$, $y_{ipk}$ is binary variable defined the job precedence:

$$y_{ipk} \begin{cases} = 1 & \text{if job } i \text{ precedes job } p \\ = 0 & \text{otherwise} \end{cases}$$

$H$ is an arbitrary positive number greater than the maximum value among all the processing times, $T_{ijk}$ of the given problem.

Although the above formulation was discussed in the context of JSPs, it can handle job recirculation without further changes. Consider equation (1), which defines intra-job precedence. Supposing two operations of the same job are processed by the same machine, the precedence which they would have to adhere to is already defined. Hence for modeling job recirculation (for our FJSP), no further constraints are required.

The JSP formulation presented above is an integer linear program. Most integer programming problems are solved by Branch & Bound (B&B) since Simplex does not return solutions for $y_{ipk}$ in integral form for equations (5) and (6). The Iterative Linear Programming with Integer Adjustments (ILPIA) technique [3] resolves this by iteratively solving the LP *twice*, hence removing the need to specifically apply integer linear programming methods. In the first pass, the set of integer linear inequalities (5) and (6) are solved by treating the binary variables ($y_{ipk}$) as real variables. Discretization on the derived $y_{ipk}$ real values is achieved by using a binary threshold function for a step response as follows. *If* $y_{ipk} > 0$, set $y_{ipk}$ to 1 *else* set $y_{ipk}$ to 0.

Although the ILPIA is more efficient than B&B, it does not guarantee optimum solutions for all instances. Good quality solutions are only found when it is applied to small problems (ie. 3 jobs 4 machine problem), and the proportion of infeasible solutions is high. As such we propose to replace ILPIA's adjustment techniques with a set of precedence defining rules. Our approach, the precedence-based ILPIA (or PB-ILPIA) iteratively solves the LP twice and uses inter-job precedence rules instead to adjust $y_{ipk}$. In the first pass, the LP formulation is the same as ILPIA. For the second pass, we define our inter-job precedence rules based on the solution of $y_{ipk}$ obtained from the first pass. Using the inter-job constraint pair from equations (5) and (6) of the first LP, we derive the algorithm given in Figure 3.

```
if ( $y_{ipk}$ == 0 ) then
        Define $S_{pk}$ precedes $S_{ik}$
else
    if ( $T_{pqk}$ - ( $H$ x $y_{ipk}$ ) > 0 ) then
        Define $S_{pk}$ precedes $S_{ik}$
    else
        Define $S_{ik}$ precedes $S_{pk}$
```

**Figure 3. Defining Inter-Job Precedence Rules**

The second LP is solved by Simplex to obtain the final solution which gives optimum values for the starting times of every operation. From experiments, PB-ILPIA in general, is able to produce less infeasible solutions from small (4x5) to medium-sized problems (10x10) as compared to canonical ILPIA. However this is not scalable as our infeasibility rate increases with larger problems. To resolve this issue, we also enforce solution feasibility as shown in Figure 4. A brief description follows. $H$ is the arbitrary constant seen in equations (5) and (6). Varying the constant $H$ affects the quality of solution produced by PB-ILPIA. Initially, $H$ is initialized to be 100 times of the longest processing time for a given chromosome. For each iteration that an infeasible solution is obtained, $H$ increases by a multiple by 10. This approach greatly reduces the percentage of infeasibility.

```
count := 0;
limit := predefined maximum of iterations
H := get_max_processing_time(chromosome)* 100
Solution := generate_with_simplex(
            chromosome, H)
while ( solution is infeasible and count <
        limit ) do
         H *= 10;
         solution := generate_with_simplex(
                     chromosome, H);
endwhile
```

**Figure 4. Enforcing solution feasibility for PB-ILPIA**

## 6.2 Adding an Operation Order Search

We use an Operation Order Search ($O^2S$), to improve the solution generated from PB-ILPIA. The Makespan Computation Algorithm adopted by GENACE [5] calculates the fitness value of each individual. While GENACE employs a pair of strings to search for a good solution, our hybrid model only utilizes a single machine assignment string; in which the operation order is determined using Simplex. Using one string reduces the search space required by the GA counterpart. We devise the following method (described in Figure 5) to further enhance the local convergence of PB-ILPIA. In Step 3, we perform random switching of operation orders by a straightforward swapping of job order within the string. A small switch in the order can produce a different schedule, while maintaining the same machine assignment.

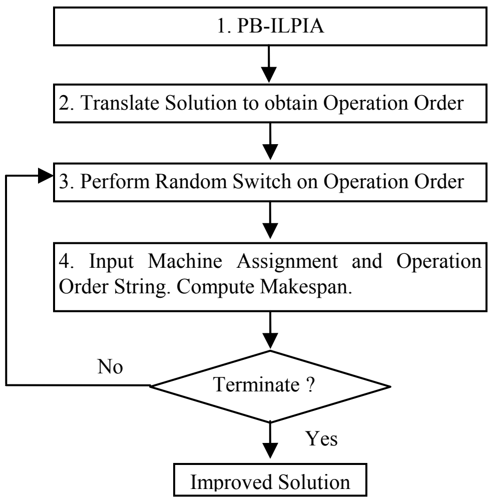
1. PB-ILPIA
2. Translate Solution to obtain Operation Order
3. Perform Random Switch on Operation Order
4. Input Machine Assignment and Operation Order String. Compute Makespan.
Terminate ?
No
Yes
Improved Solution

**Figure 5. Adding Operation Ordered Search to PB-ILPIA**

## 6.3 Integration with the Genetic Algorithm

We now describe how our local optimization method (PB-ILPIA with $O^2S$ ) is integrated with a GA. *The main idea is to translate an individual from the GA population into a LP formulation to be solved by PB-ILPIA with $O^2S$.* We adopt the simple and straightforward string $C_2$ from [13] using real number encoding as shown below.

| $O_{11}$ | $O_{12}$ | … | $O_{ij}$ | … | $O_{nl}$ |
|---|---|---|---|---|---|
| $f_{idx}\left(M_{O_{11}}\right)$ | $f_{idx}\left(M_{O_{12}}\right)$ | … | $f_{idx}\left(M_{O_{ij}}\right)$ | … | $f_{idx}\left(M_{O_{nl}}\right)$ |

where $M_{O_{ij}}$ is the machine to perform operation $O_{ij}$, $M_{O_{ij}} \in M_{ij}$, and $f_{idx}\left(M_{O_{ij}}\right)$ gives the set of index numbers of available machines for $O_{ij}$.

The translation process for the LP generator is as follows.

1. First Operation: For every $O_{ijk}$, such that $j$ =1, translate into equation (2).
2. Intra Job Precedence: For $L$ operations in the job $i$, generate equation (1) such that $O_{ij\text{-}1k}$ precedes $O_{ijh}$ for $1 < j \leq L$, where $t_{ij\text{-}1k}$ is processing time referenced from the machine $k$ encoded under $O_{ij\text{-}1}$ in the string.
3. Inter Job Precedence: For every $O_{ijk}$ of job $i$, scan all operations in subsequent jobs $p$ where $i < p \leq N$ , if the same machine $k$ is assigned to $O_{ij}$ and $O_{pq}$, then generate equations (5) and (6), where $T_{ijk}$ and $T_{pqk}$ are respective processing times referenced from machine $k$ in accordance with the problem.
4. Objective Function: We consider the minimization of all start times for the last operation of every job.

This process produces the initial LP instance to be solved by PB-ILPIA with $O^2S$ described previously.

## 7. EMPIRICAL RESULTS

This Section empirically evaluates some of the components featured in the algorithm. The ALPINE algorithm was coded in C++ on a 3-GHz Pentium IV. The experiments were conducted using a population of 100 individuals, where each run consists of 200 generations unless otherwise specified. The following parameters are used: crossover 0.75, mutation 0.2, replacement 0.1, number of generations to perform learning 5, number of chromosomes used to influence operation belief space 10. Test cases used are benchmarks problems by Kacem *et al.* [8].

## 7.1 Number of Niches

This experiment compares the efficiency of the ALPINE algorithm with respect to a predefined number of niches. We run the test using a 4x5 FJSP. The following parameters are used to measure the niche performance: average total evaluations of individuals to obtain its makespan during the whole process $C_{avg}$, and number of predefined niches, $N$. The following results were compiled after 20 runs.

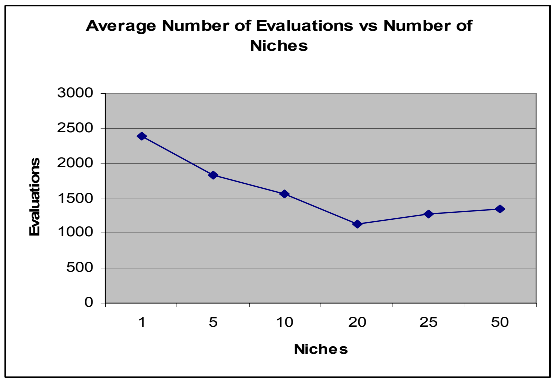

**Figure 6. Average Total Evaluations vs. Number of Niches**

From Figure 6, we observe a general decrease in the average number of evaluations ($C_{avg}$ ) from $N$ = 1 to $N$ = 20, and an increase till $N$ = 50. When $N$ is small, the neighborhood radius is larger. This accounts for more members in a niche. Hence during the intensification phase, more evaluations are required. Likewise for the case when $N$= 50, while there are less members per niche for evaluation, approximately 50 niche representatives will have to be evaluated for each generation. This value is high since it is approximately half of the population, which defeats our objective to reduce the number of evaluations in the first place. Hence we must carefully account for the tradeoffs to achieve maximum efficiency. For our experiment, we achieve efficient results when $N$ = 20 for a population of 100 individuals. For problem conditions that require a larger initial population, we can use the same ratio of niche to population size above, that is 0.2, to maintain the same efficiency.

## 7.2 CDR-PopGen, PB-ILPIA with $O^2S$ Performance

In this experiment, evaluation is done for 200 individuals using the respective permutation in each FJSP, where $M_{min}$ is the minimum makespan, $M_{avg}$ is the average makespan and $S_{dev}$ is the standard deviation. As observed from Table 1, $O^2S$ effectively reduces the average makespan and achieves better minimum makespans for all cases. For the latter, the minimum makespans for 4x5 and 10x10 instances with CDR-PopGen [5] remain unchanged with the use of $O^2S$. This is in line with our observation that PB-ILPIA already produces near optimal solutions. In the area where PB-ILPIA is deficient in producing

**Table 1. Performance of Combinatorial Search Methods**

| | 4x5 | | | 8x8 | | | 10x7 | | | 10x10 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $M_{min}$ | $M_{avg}$ | $S_{dev}$ | $M_{min}$ | $M_{avg}$ | $S_{dev}$ | $M_{min}$ | $M_{avg}$ | $S_{dev}$ | $M_{min}$ | $M_{avg}$ | $S_{dev}$ |
| *PB-ILPIA* | 18 | 36.97 | 19.07 | 36 | 60.66 | 11.29 | 36 | 72.80 | 31.85 | 32 | 51.54 | 11.6 |
| *PB-ILPIA+O²S* | 15 | 31.81 | 18.84 | 34 | 48.59 | 9.34 | 28 | 62.57 | 32.20 | 26 | 41.55 | 9.22 |
| *CDR,PB-ILPIA* | 11 | 12.02 | 0.86 | 17 | 23.34 | 4.09 | 13 | 18.94 | 3.46 | 8 | 10.59 | 1.02 |
| *CDR,PB-ILPIA+O²S* | 11 | 11.90 | 0.59 | 14 | 18.29 | 2.05 | 11 | 14.71 | 1.78 | 8 | 8.58 | 0.81 |

*CDR: Initialization by Composite Dispatch Rule, $O^2S$: Operation Order Search*

good quality solutions, the addition of $O^2S$ helps to improve the overall solution. This is illustrated in Table 2 which shows the overall average decrease in makespan with the addition of $O^2S$. Further decrease in average makespan is observed with the use of CDR-PopGen, which suggests that good machine assignments are essential for producing good schedules. The large reduction in $S_{dev}$ (from 18.84 to 0.59 for the 4x5 instance) shows that CDR-PopGen is very effective in pruning the search space. One point worth noting is the figures from CDR-PopGen in two instances (8x8 and 10x7) where approximately half the deviation is achieved with $O^2S$, along with better minimum makespan. A smaller deviation with a reduced mean shows the potential of the population to produce good quality solutions. Hence CDR-PopGen is indeed very capable of producing near optimal, if not optimal schedules. However this can be only achieved with a good operation string. The optimal operation string search is achieved by PB-ILPIA with $O^2S$. In conclusion, the three methods are complementary to form a robust local search method.

**Table 2. Proportional Improvement with $O^2S$**

| | Without *CDR* | *CDR* |
|---|---|---|
| 4x5 | 0.154 | 0.009 |
| 8x8 | 0.190 | 0.202 |
| 10x7 | 0.155 | 0.207 |
| 10x10 | 0.184 | 0.183 |

**Table 3. Comparison of best makespans**

| | *GENACE* | *ALPINE* |
|---|---|---|
| 4x5 | 11 | 11 |
| 8x8 | 16 | 14 |
| 10x7 | 13 | 11 |
| 10x10 | 9 | 8 |

Table 3 compares the best makespan achieved with CDR-PopGen between GENACE and ALPINE. The improvement in three test instances suggests that a good operation order search may be required to further enhance CDR-PopGen.

## 7.3 Success Rate Comparison

This experiment compares the efficiency of the ALPINE algorithm to locate the known global optimum for 8x8 and 10x7 instance. We use GENACE as our basis for comparison. Our results are obtained using PB-ILPIA with $O^2S$. CDR-based population initialization is done for 80 individuals. As seen in Table 4, ALPINE has obtained the known global optima within 200 generations, and in the 10x7 instance; achieved a new global optima undiscovered by GENACE. Two other points are worth noting. Firstly, the low number of average generations required for ALPINE can be explained by the fact that in some trials, such as the one in 8x8 instance, global optima is already obtained by CDR-PopGen alone. However, in comparison with GENACE, CDR-PopGen produces inferior results. This shows that CDR-PopGen cannot produce better results than when coupled with PB-ILPIA and $O^2S$. Hence, while CDR-PopGen is able to find good machine assignments, a good operation order search is still needed to produce optimal results. Secondly, a higher success rate is observed for ALPINE. When CDR-PopGen fails to produce an optimal solution, as in the case of the 10x7 instance, ALPINE is able to complement this by locating the optimum's basin through a global genetic search. Unlike GENACE, it is less likely to be trapped in a local optima, thus showing the efficiency of our niched-based hybrid structure. Another reason being the search space explored in our algorithm is much smaller since only one machine assignment string is needed, leading to higher success rate as well.

## 7.4 Comparisons with benchmark problems

Table 5 shows how ALPINE fares against other algorithms. The following benchmarks are used. Temporal Decomposition, Classical Genetic Algorithm, Approach by Localization, Localization and Controlled Evolutionary Algorithm (AL + CGA) referenced from Kacem *et al.*[7]. Hybridization of Fuzzy Logic and Evolutionary Algorithm (FL+EA) by Kacem *et al.*[8]. Hybrid Particle Swarm Optimization and Simulated Annealing (PSO+SA) by Xia *et al.* [15] and GENACE [5], a cultural evolution algorithm.

Overall, ALPINE compares well for all test cases. In particular, a new global optima has been found by ALPINE for the 10x7 instance. However, only a sub-optimal solution can be obtained for the 15x10 instance. This is due to the deficiency of PB-ILPIA to produce good solutions with increasing problem size (discussed in Section 5), hence $O^2S$ becomes less effective in finding the global optima. Nevertheless a near optimal solution is obtained with a small deviation. It is however worth noting that ALPINE is highly efficient for medium-sized problems and below, where an optimal solution can be obtained by simply applying CDR-PopGen with PB-ILPIA and $O^2S$, without the need for a global genetic search.

## 8. CONCLUSION AND FUTURE WORK

In this paper, we presented a hybrid evolutionary algorithm called ALPINE for globalized niche-based evolution with localized Simplex search. It combines favorable machine assignments by CDR-PopGen [5], near-optimal operation order derived by PB-ILPIA, improvement of the latter with randomized $O^2S$ and removal of local optima through the parallelism of GA. While each feature of ALPINE is insufficient alone to locate the global optima, a hybrid of the above methods demonstrates synergism. Experimental results on Kacem's benchmarks problems using our hybrid have shown that ALPINE outperforms many existing hybrid techniques and is on par with GENACE, a cultural

**Table 4.** **Results of comparison of efficiency in locating global optima**

| | GENACE | | | | ALPINE | | | |
|---|---|---|---|---|---|---|---|---|
| | $M_{best}$ | $M_{avg}$ | $Rs$ | $G_{avg}$ | $M_{best}$ | $M_{avg}$ | $Rs$ | $G_{avg}$ |
| 8 x 8 | 14 | 14.50 | 60 | 45.17 | 14 | 14 | 100 | 0.10 |
| 10x7 | 12 | 13.80 | 0 | 0.00 | 11 | 11.3 | 70 | 6.71 |

$G_{avg}$ : Average generations to reach optimum *Rs : Success rate of obtaining optimum(%)*

*$M_{avg}$: Average makespan obtained* *$M_{best}$: Best makespan obtained*

**Table 5.** **Results of comparison with benchmark problems**

| | Type | Temporal/D | Classic GA | By Localization | AL+CGA | FL+EA | PSO+SA | GENACE | ALPINE |
|---|---|---|---|---|---|---|---|---|---|
| 4x5 | T | - | - | - | - | 16 | - | 11 | 11 |
| 8x8 | P | 19 | 16 | 16 | 15 | - | 15 | 14 | 14 |
| 10x7 | T | - | - | - | - | 15 | - | 12 | 11 |
| 10x10 | T | 16 | 7 | 8 | 7 | 7 | 7 | 7 | 7 |
| 15x10 | T | - | - | - | 23 | 23 | 12 | 12 | 13 |

*Type : Total (T) or Partial (P) flexibility.*

evolution algorithm. A new global optima for the 10x7 FJSP has also been found.

The proposed $O^2S$ used in this paper consists of a blind random search. Further investigation can be done to improve the operation order by determining the bottleneck machines or jobs. Larger problems can be tackled by dividing them into smaller problems, where the start times of the subproblems can be easily added as a constraint. Breaking the problem into suitable sizes allows our hybrid method to operate efficiently. A clear bottleneck in the algorithm to tackle large problems is the Simplex method. While it provides good solutions for small problems, a more efficient alternative may be needed for larger operational orders.

## 9. REFERENCES

[1] Branke, J., “Memory enhanced evolutionary algorithms for changing optimization problems,” *Proc. IEEE Congress on Evolutionary Computation*, pp. 1875-1882, 1999.

[2] Chelouah, R. and Siarry, P., “Genetic and Nelder–Mead algorithms hybridized for a more accurate global optimization of continuous multiminima functions,” *European Journal of Operational Research*, vol. 148, pp. 335-348, 2003.

[3] Foo, Y. S. and Takefuji, T., “Integer linear programming neural networks for job-shop scheduling,” *IEEE International Conference on Neural Networks*, vol. 2, pp 341-348, 1988.

[4] Gonçalves, J. F., Mendes, J. J. M., Resende, M. G. C., “A hybrid genetic algorithm for the job shop scheduling problem,” *European Journal of Operational Research*, vol. 167(1), pp. 77-95, 2005.

[5] Ho, N. B. and Tay, J. C., "GENACE: An Efficient Cultural Algorithm for solving the Flexible Job-Shop Problem," *Proc. IEEE Congress on Evolutionary Computation*, pp. 1759-1766, 2004.

[6] Jain, A. S. and Meeran, S., “Job Shop Scheduling using Neural Networks,” *International Journal of Production Research*, vol. 36(5), pp 1249-1272, 1998.

[7] Kacem, I., Hammadi, S., Borne, P., “Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 32(1), pp. 1-13, 2002.

[8] Kacem, I., Hammadi, S., Borne, P., “Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic” *Mathematics and Computers in Simulation*, vol. 60, pp. 245-276, 2002.

[9] Kacem, I., “Genetic algorithm for the flexible job-shop scheduling problem,” *Proc. IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, pp. 3464-3469, 2003.

[10] Mathias, K. E., Whitley, L. D., Stork, C., Kusuma, T., “Staged hybrid genetic search for seismic data imaging,” *Proc. The First IEEE Conference on Evolutionary Computation*, vol.1, pp. 356-361, 1994.

[11] Mesghouni, K., Hammadi, S., Borne, P., “Evolution programs for job-shop scheduling,” *Proc. IEEE International Conference on Computational Cybernetics and Simulation*, vol. 1, pp. 720-725, 1997.

[12] Panek, S., Stursberg, O., Engell, S., “Optimization of timed automata models using mixed integer programming,” *Formal Modeling And Analysis of Timed Systems: First International Workshop*, vol. 2791, pp. 73-87, 2004.

[13] Tay, J. C. and Wibowo, D., "An Effective Chromosome Representation for Evolving Flexible Job Shop Schedules," *Genetic and Evolutionary Computation Conference*, vol. 2, pp 210-221, 2004.

[14] Wei, L. and Zhao, M., “A niche hybrid genetic algorithm for global optimization of continuous multimodal functions,” *Applied Mathematics and Computation*, vol. 160, pp. 649-661, 2005.

[15] Xia, W. and Wu, Z., “An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems” *Computers & Industrial Engineering*, vol. 48(2), pp. 409-425, 2005.

[16] Yen, J., Liao, J. C., Bogju, L., Randolph, D., “A hybrid approach to modeling metabolic systems using a genetic algorithm and simplex method,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28 , pp. 173-191,1998.