ICEC '98 (1998 IEEE International Conference on Evolutionary Computation) pp.230–234

Solving the C_{sum} Permutation Flowshop Scheduling Problem by Genetic Local Search

Takeshi Yamada, NTT Communication Science Labs., Kyoto, JAPAN
Colin R. Reeves, Coventry University, Coventry, UK

Abstract— **In this paper a new metaheuristic method is proposed to solve the classical permutation flowshop scheduling problem with the objective of minimizing sum of completion times. The representative neighbourhood combines the stochastic sampling method mainly used in Simulated Annealing and the best descent method elaborated in Tabu Search and integrates them naturally into a single method. The method is further extended into the Genetic Local Search framework by using a population and a special crossover operator called multi-step crossover fusion. Computational experiments using benchmark problems demonstrate the effectiveness of the proposed method.**

Keywords— **flowshop scheduling, genetic algorithms, tabu search, stochastic sampling, path relinking**

1. Introduction

In the last two decades, new approximation methods such as Simulated Annealing (SA), Genetic Algorithms (GA) and Tabu Search (TS) have been successfully applied to combinatorial optimization problems for which classical methods such as Branch and Bound did not work effectively. These methods, which were previously studied in different communities and different contexts, are now collectively called Meta-Heuristics (MH) to be studied in a single framework. While sharing the core features in common, they have different characteristics such as SA's stochastic sampling, GA's population-based search strategy and TS's adaptive memory. It is natural to assume that a very powerful algorithm can be constructed if one can integrate these characteristics together into a single algorithm, because they are not mutually exclusive but rather compatible.

In this paper, we propose one approach to construct such a powerful unified algorithm to solve the permutation flowshop scheduling problem (PFSP). In Section 2, the flowshop scheduling problem minimizing sum of completion times is defined and its neighbourhood structure is discussed in Section 3. Using this neighbourhood structure, a stochastic local search algorithm is proposed as a base algorithm in Section 4, and then this algorithm is extended to include TS in Section 5 and then GA or Genetic Local Search (GLS) in Section 6. Experimental results using Taillard's benchmark problems are shown in Section 7.

This work was undertaken when the first author was a visiting researcher in the School of Mathematical and Information Sciences, Coventry University, UK

2. The permutation flowshop scheduling problem

The permutation flowshop scheduling problem (PFSP) treated in this paper is often designated by the symbols $n/m/P/Obj$, where n jobs have to be processed on m machines in the same order. P indicates that only permutation schedules are considered, where the order in which the jobs are processed is identical for all machines. Hence a schedule is uniquely represented by a permutation of jobs. Obj describes the performance measure by which the schedule is to be evaluated—the objective function. For example, $n/m/P/C_{max}$ is the problem of minimizing the makespan C_{max}, while $n/m/P/C_{sum}$ is the problem of minimizing the sum of the completion times C_{sum}.

We have already proposed an efficient method to solve the $n/m/P/C_{max}$ in [1], [13]. In this paper we deal with the $n/m/P/C_{sum}$. Compared to the C_{max} problem, the C_{sum} problem is more difficult to optimize, mainly because the calculation of the objective function is more time consuming, and problem specific knowledge such as critical blocks cannot be used. Only some constructive algorithms based on heuristic rules such as LIT and SPD are known in the literature [2]. Recently J.Liu proposed a new constructive method based on, and consistently better than, LIT and SPD [3].

3. Neighbourhood structure

A neighbourhood $N(x)$ of a point x in a search space can be defined as a set of new points that can be reached from x by exactly one transition or move (a single perturbation of x). One of the well-known transition operators for PFSP is the *shift operator* which takes a job from its current position and re-inserts it in another position. The neighbourhood of a schedule s induced by the shift operator is the set of all schedules obtained from s by the operator. For each of n jobs, there are $n-1$ possible positions to be re-inserted; the size of this neighbourhood is thus $n \times (n-1)$, which is rather too big for even moderate-sized problems.

3.1. Representative neighbourhood

Nowicki and Smutnicki originally proposed the *representative* neighbourhood method for the C_{max} problem[4]. In their approach, a reduced neighbourhood is generated from the original neighbourhood by clustering its members and choosing the best from each cluster as a representative. We adopt this method for the C_{sum} problem as shown below.

Let s be a job sequence of the current solution, and $s[i,k]$ be a new job sequence obtained from s by moving a job from the i^{th} position in s and re-inserting it in the k^{th} position. $N_i^a(s)$ and $N_i^b(s)$, subsets of $N(s)$, are defined as follows:

$$N_i^a(s) = \{s[i,k] \mid i < k \le n\}, N_i^b(s) = \{s[i,k] \mid 1 \le k < i\}.$$

Thus the original neighbourhood $N(s)$ is divided into clusters consisting of $N_i^a(s)$ and $N_i^b(s)$. Let $\overline{N_i^a(s)}$ and $\overline{N_i^b(s)}$ be one of the best members in $N_i^a(s)$ and $N_i^b(s)$ respectively. The representative neighbourhood $N^*(s)$ can be denoted as:

$$N^*(s) = \{\overline{N_i^a(s)} \mid 1 \le i < n\} \cup \{\overline{N_i^b(s)} \mid 1 < i \le n\}.$$

In most local search algorithms, the NS operators to choose a new member from the neighbourhood of the current solution can be categorized into two types according to their choice criteria [5] : one is best descent, and the other is first descent. The best descent method scans all the members in the neighbourhood at once and choose the best as a new current solution. This is suitable when the neighbourhood size is small and the cost of evaluating all the members is negligible. Tabu search can be seen as an extension of this method. The first descent method selects one member (at random) and accepts it if it is sufficiently good, otherwise selects another one. This can be used even when the neighbourhood size is large. The stochastic sampling in stochastic local search including SA can be seen as an extension of this method. The representative neighbourhood fills the gap between these two criteria: a cluster $N_i(s)$ is chosen randomly by using first descent, then best decent is applied to evaluate all the members in $N_i(s)$ of which the best is chosen as a representative. Figure 1 illustrates this process. As we will see in the later section, this enables the TS and a stochastic local search method to integrate into a single unified method.

4. Local search

This section and the next two sections are devoted to a construction of an algorithm to solve the C_{sum} permutation flowshop scheduling problem effectively. Here we start from a simple local search method and then it will be extended to a more powerful algorithm in the later sections.

The base local search algorithm is constructed as a stochastic method which is similar to SA, where a member y from the neighbourhood $N(x)$ of the current solution x is selected at random and tested if it is good enough to be accepted as a new current solution according to the Metropolis

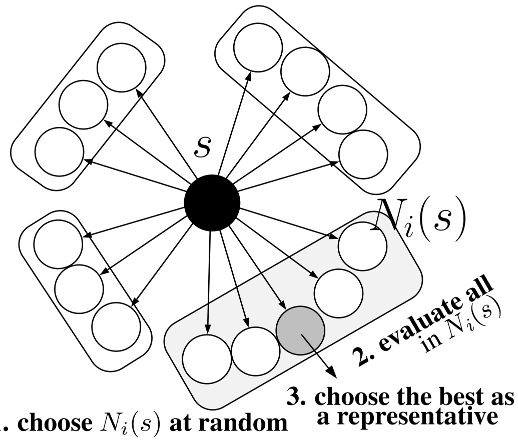

Fig. 1. Representative neighbourhood

criterion. That is, $y \in N(x)$ is accepted with probability 1 if $V(y) < V(x)$, where $V(\cdot)$ is the objective function to be *minimized*, otherwise with probability

$$P_T(y) = exp(-\Delta V/T), \text{where } \Delta V = V(y) - V(x)\ . \quad (1)$$

Here P_T is called the *acceptance probability*. In SA, the parameter T (called the *temperature*) decreases to zero following an annealing schedule as the number of iterations increases. A very long time is required for SA to converge to a high-quality solution, so a more practical approach is to keep T constant: $T = c$. However c is problem dependent, and it is difficult to predict an appropriate value. The outline of this algorithm is described in Figure 2.

In our stochastic local search, T is determined adaptively by an observed value called *uphill ratio* r_{up} [6]. r_{up} is a ratio of the number of recently accepted non-improving solutions over the number of recently accepted both improving and non-improving solutions. Given a constant R_{up} as a target uphill ratio (for example $R_{up} = 0.25$), we try to keep the difference between r_{up} and its target value R_{up}: $|r_{up} - R_{up}|$ as small as possible throughout the search by adjusting T adaptively. To be more concrete: assuming that r_{up} is not very different from R_{up} (i.e R_{up}/r_{up} is within a reasonable range), we update T as $T = T * (R_{up}/r_{up})$ when $|r_{up} - R_{up}| > \epsilon$ (ϵ is a small constant, 0.01 for example).

5. Tabu search

The simple stochastic local search algorithm described above does not make use of memory, either short term or long term, as Tabu Search (TS) does. The use of the representative neighbourhood makes it easy to have tabu-list style adaptive memory as in the TS.

If a solution $s[i,k]$ generated from the current solution s by moving a job $j = s[i]$ to the k^{th} position is accepted, the pair (j,i), i.e. the job and its original position, is stored on the top of a list of length l and recorded as *tabu*. The

- Select a starting point: $x = x_{best} = x_0$.

do

  do

  - Select a point $y \in N(x)$ at random.
  - Accept y with probability 1 if $V(y) \leq V(x)$, and with probability $P_T(y)$ otherwise.

  until y is accepted.

- Set $x = y$.
- If $V(x) < V(x_{best})$ then set $x_{best} = x$.

until some termination condition is satisfied.

Fig. 2. Stochastic local search

oldest element in the list is then deleted. In subsequent iterations, a solution generated by moving job j to the i^{th} position should not be accepted as long as (j, i) is on the list. In our representative neighbourhood scheme, this is achieved by excluding the tabu solution from the calculation of the representative best solution ($\overline{N_i^a(s)}$ and $\overline{N_i^b(s)}$) so that the representative neighbourhood will contain no tabu solutions. Because the tabu solutions have already been excluded from the representative neighbourhood, there is no need to modify the stochastic local search procedure described in Section 4 by this modification.

6. Genetic local search

The local search method described above is further improved by embedding it into the framework of Genetic Local Search (GLS) or population-based local search. There is a set of P agents each of which performs short term multi-start local search: each local search terminates after a fixed number of iterations L_1 and the best solution obtained is used as an initial starting point of the next local search. The short term local search method used here is the one described in Section 4 and 5. The agents occasionally exchange information and the information exchange has two forms: one is called *selection* which passes the solution information of a successful agent to the other members in the population, and the other is called *crossover* which recombines solutions from more than one agent in order to obtain new solutions.

Algorithm 3 describes the outline of the GLS routine, where a steady state model is used as a selection model. In this model, the population is ranked according to the C_{sum} values, and two solutions are selected as parents from the population with a probability inversely proportional to their ranks. The algorithm would use crossover on the parents (or use mutation on the first parent if the parents are too close to each other) with probability P_X (P_X = 0.5 for example) and generate a new solution, otherwise stochastic local search is used on the first parent. The newly generated solution q is inserted into the population only if its C_{sum} is better than the worst in the population. To avoid premature convergence even under a small-population condition, if an individual with the same C_{sum} value already exists in the population, then q is not inserted into the population in Step 7.

1. Initialize population: randomly generate a set of P permutation schedules. Sort the population members in descending order of their C_{sum} values.
2. Repeat Step 3 to Step 7 until some termination condition is satisfied.
3. Select two schedules p_1, p_2 from the population with a probability inversely proportional to their ranks.
4. Do Step 5 with probability P_X, or otherwise do Step 6.
5. **If** the distance between p_1, p_2 is less than d_{min}, apply **mutation** to p_1 and generate q. **Otherwise**, apply **crossover** to p_1, p_2 and generate q.
6. Apply **local search** and generate q.
7. If q's C_{sum} value is less than the worst in the population, and no member of the current population has the same C_{sum} as q, replace the worst individual with q.
8. Output the best member in the population and terminate.

Fig. 3. Genetic local search for the PFSP

The termination condition can be given, for example, as a fixed number of iterations L_2. The local search algorithm used is the one described in Section 4. The crossover and mutation operators are discussed in the next subsection.

6.1. Multi-step crossover fusion

As has been observed elsewhere (see, for example, [7]), traditional genetic crossover actually has two functions, which we denote by F1 and F2. Firstly (F1) it focuses attention on a region between the parents in the search space; secondly (F2), it picks up possibly good solutions from that region. Unlike traditional crossover operators, Multi-step crossover fusion (MSXF) is a more search oriented: it is designed as an extension of local search algorithm described in section 4, but has the functions F1 and F2, so well call it as *crossover* here. In this section we review MSXF only briefly. Please refer to [8], [9], [1], [10], [13] for more detail.

MSXF is defined in a problem-independent manner using a neighbourhood structure and a distance measure, both of which are very common for most combinatorial optimization problems. Let the parent solutions be p_1 and p_2, and let the distance between any two individuals x and y be $d(x, y)$. A short term local search is carried out starting from p_1 and using p_2 as a reference point as follows. First

x is set to p_1. All members in $N(x)$ are sorted in ascending order of $d(y_i, p_2)$ so that $y_i \in N(x)$ with a smaller index i has a smaller distance $d(y_i, p_2)$. One of the members $y_i \in N(x)$ is selected with a probability inversely proportional to the index i. Then y_i is accepted according to the Metropolis criterion in Equation (1). As previously suggested, the termination condition can be given by, for example, a fixed number of iterations L_1 The best solution q is used in the next generation (see Figure 4).

- Let p_1, p_2 be the relevant solutions.
Set $x = q = p_1$.
do
  - For each $y_i \in N(x)$, calculate $d(y_i, p_2)$.
  - Sort $y_i \in N(x)$ in ascending order of $d(y_i, p_2)$.

  do
    - Select y_i from $N(x)$ with a probability inversely proportional to the index i.
    - Calculate $V(y_i)$ if $V(y_i)$ is unknown.
    - Accept y_i with probability 1 if $V(y_i) \leq V(x)$, and with probability $P_T(y_i)$ otherwise.
    - Change the index of y_i from i to n, and the indices of y_k ($k \in \{i+1, \ldots, n\}$) from k to $k-1$.

  until y_i is accepted.
- Set $x = y_i$.
- If $V(x) < V(q)$ then set $q = x$.

until some termination condition is satisfied.
- q is used in the next generation.

Fig. 4. Multi step crossover fusion

In contrast with the stochastic local search described in Figure 2, MSXF carries out a short term *navigated* local search starting from one of the parent solutions to find new good solutions, where the other parent is used as a reference point so that the search direction is biased toward the parent. Therefore the stochastic local search is a fine-grained search around p_1, whereas MSXF performs more coarse-grained search but in more wider area between p_1 and p_2. It should be noted that crossover and local search are treated equally in Figure 3, especially when $P_X = 0.5$. The proposed method makes use of the combination of these two search methods of different focus and preliminary experiments suggests that the value $P_X = 0.5$ is appropriate for our problem.

It is pertinent to note that the approach of generating solutions from search paths joining parent solutions was first proposed in the scatter search method of Glover [11], which also included the strategy of using heuristics to improve the offspring (as later incorporated in Guided Local Search). The use of neighborhood spaces as the basis for such search paths was proposed in the extension of scatter search called path relinking, as described in the context of tabu search in Glover and Laguna [12]. MSXF may equally well be viewed from the perspective of path relinking (see [13]).

6.2. Multi-step mutation fusion

Mutation is nearly always regarded as an integral part of a GA. In our case, we make use of mutation only when parents are too close to each other. In such circumstances, MSXF is not applicable as the number of possible neighbours is severely curtailed—in the limit, where the distance is just one move, there is of course no path at all. In such cases (defined to be when the distance was less than a value d_{min}), a mutation operator called *Multi-Step Mutation Fusion* (MSMF) is applied with the aim of diversifying the search. MSMF can be defined in the same manner as MSXF except that the neighbours of x are sorted in *descending* order of $d(y_i, p_2)$, and the most *distant* solution is stored and used in the next generation instead of q, if q does not improve the parent solutions. From a path relinking viewpoint, we are attempting to extrapolate the path between the solutions, rather than (as normal) to interpolate it.

7. Experimental Results

We applied our method to some of Taillard's benchmark problems (ta problems, in short) [14]. First it was applied to relatively easy problems from ta001 to ta030 (the number of jobs is 20 and the number of machines is 5, 10 and 20: denoted by 20x5, 20x10 and 20x20). Six runs were carried out for each problem with different random seeds. The parameters used in these experiments are: $P = 5, L_1 = 1000, L_2 = 700, P_X = 0.5$ and the length of the tabu list $l = 7$.

Here quite consistent results were obtained, i.e. almost all of the 6 runs converged to the same job sequence in a short time (from a few seconds to a few minutes) before the limit of $L_2 = 700$ was reached on a HP workstation. The best results (and they are also the average results in most cases) are reported in Table I together with the results obtained by the constructive method (NSPD) due to J.Liu [3].

Problems ta031 to ta050 (50x5 and 50x10 problems) are much more difficult and the best results were different in each run. Ten runs were carried out for each problem with different random seeds. The parameters used in these experiments were: $P = 30, L_1 = 10000, L_2 = 700, P_X = 0.5$. It takes 45 minutes per run for 50x5 problems (ta031 to ta040) and 90 minutes for 50x10 problems (ta041 to ta050).

It is difficult to say how good these solutions are, in other words, how far they are from the global optima. Even for the easier problems in Table I, there is no guarantee that the best solutions obtained so far are optimal, although we believe that they are at least very close to being so. For the problems in Table II, it is almost certain that our best results are not optimal. In fact we found one solution of

TABLE I
Taillard's benchmark results (ta001 – ta030)

prob	best	NSPD	prob	best	NSPD	prob	best	NSPD
001	14033	14281	011	20911	21520	021	33623	34119
002	15151	15599	012	22440	23094	022	31587	32706
003	13301	14121	013	19833	20561	023	33920	35290
004	15447	15925	014	18710	18867	024	31661	32717
005	13529	13829	015	18641	19580	025	34557	35367
006	13123	13420	016	19245	20010	026	32564	33153
007	13548	13953	017	18363	19069	027	32922	33763
008	13948	14235	018	20241	21048	028	32412	33234
009	14295	14552	019	20330	21138	029	33600	34416
010	12943	13054	020	21320	22212	030	32262	33045

TABLE II
Taillard's benchmark results (ta031 – ta040)

prob	best	average	NSPD	prob	best	average	NSPD
031	64860	64934.8	66590	041	87430	87561.4	90373
032	68134	68247.2	68887	042	83157	83305.8	86926
033	63304	63523.2	64943	043	79996	80303.4	83213
034	68259	68502.7	70040	044	86725	86822.4	89527
035	69491	69619.6	71911	045	86448	86703.7	89190
036	67006	67127.6	68491	046	86651	86888.0	91113
037	66311	66450.0	67892	047	89042	89220.7	93053
038	64412	64550.1	66037	048	86924	87180.5	90614
039	63156	63223.8	64764	049	85674	85924.3	91289
040	68994	69137.4	69985	050	88215	88438.6	91622

$C_{sum} = 64803$ for problem ta031 by an overnight run.

8. Conclusions

A new genetic local search method is proposed to solve the C_{sum} permutation flowshop scheduling problem. This method effectively integrates the stochastic sampling of Simulated Annealing, the adaptive memory using the tabu list of Tabu Search and the population-based search of Genetic Algorithms into a single unified framework as summarized in Figure 5. The method is applied to Taillard's benchmark problems. Experimental results demonstrate the effectiveness of the proposed method.

References

[1] T. Yamada and C.R.Reeves, "Permutation flowshop scheduling by genetic local search," in *2nd IEE/IEEE Int. Conf. on Genetic ALgorithms in Engineering Systems (GALESIA '97)*, pp. 232–238, 1997.

[2] C. Wang, C. Chu, and J. Proth, "Heuristic approaches for $n/m/F/\sum C_i$ scheduling problems," in *European Journal of Operational Research*, pp. 636–644, 1997.

[3] J. Liu, "A new heuristic algorithm for csum flowshop scheduling problems." Personal Communication, 1997.

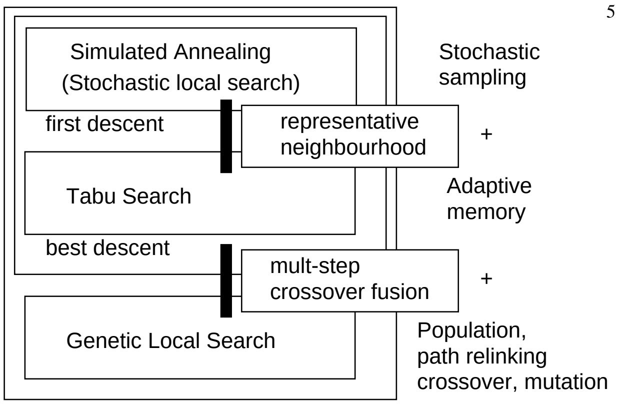

Fig. 5. The framework of the proposed method

[4] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for the permutation flow-shop problem," *European Journal of Operational Research*, vol. 91, pp. 160–175, 1996.

[5] T. Yamada and R. Nakano, "A genetic algorithm with multi-step crossover for job-shop scheduling problems," in *1st IEE/IEEE Int. Conf. on Genetic ALgorithms in Engineering Systems (GALESIA '95)*, pp. 146–151, 1995.

[6] T. Yamada, B. Rosen, and R. Nakano, "A simulated annealing approach to job shop scheduling using critical block transition operators," in *Proc. IEEE Int. Conf. on Neural Networks*, pp. 4687–4692, 1994.

[7] C. Höhn and C. R.Reeves, "Are long path problems hard for genetic algorithms?" in *4th International Conference on Parallel Problem Solving from Nature*, pp. 134–143, 1996.

[8] T. Yamada and R. Nakano, "Scheduling by genetic local search with multi-step crossover," in *4th International Conference on Parallel Problem Solving from Nature*, pp. 960–969, 1996.

[9] T. Yamada and R. Nakano, "A fusion of crossover and local search," in *IEEE International Conference on Industrial Technology (ICIT '96)*, pp. 426–430, 1996.

[10] T. Yamada and R. Nakano, *Job Shop Scheduling. Chapter 7 in A.M.S. Zalzala and P.J.Fleming (Ed.) Genetic algorithms in engineering systems*. The Institution of Electrical Engineers, London, UK, 1996.

[11] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision Science*, vol. 8, pp. 156–166, 1997.

[12] F. Glover and M. Laguna, *Tabu Search. Chapter 3 in C.R.Reeves (Ed.) Modern Heuristic Techniques for Combinatorial Problems*. (Recently re-issued (1995) by MacGraw-Hill, London: Blackwell Scientific Publications, Oxford, 1993.

[13] C. R. Reeves and T. Yamada "Genetic Algorithms, Path Relinking and the Flowshop Sequencing Problem", *special issue on scheduling, Evolutionary Computation*, MIT press, Cambridge MA, 1998 (to appear).

[14] E. Taillard, "Benchmarks for basic scheduling problems," *E. J. of Oper. Res.*, pp. 278–285, 1993.