Project 4: Solving Job Shop Scheduling Problem Using Bio-Inspired Algorithms

Per Magnus Veierland
permve@stud.ntnu.no

April 18, 2017

SCHEDULE REPRESENTATION

All three optimizers utilize the same schedule representation as shown in Figure 1. This representation is able to represent both infeasible and feasible solutions to the Job-Shop Scheduling Problem (JSSP). Depending on context in the implementation, this representation either represents the job preference for each machine in a solution, or it represents a solution to the JSSP directly. For both the Particle Swarm Optimization (PSO) and the Bees Algorithm (BA), random individuals are generated as part of the search. This is done by building a schedule with a randomly permuted job ordering for each machine. If treated as a solution directly, such a solution may be infeasible as machines can deadlock. To ensure that all created solutions are feasible, each possibly infeasible solution is "developed" by treating the representation as a preference description. The G&T algorithm as described in [1] details how a preference description can be used to generate an active schedule which must be feasible.

The G&T algorithm can be described in four steps:

1. Initialize schedule $S = \varnothing$; Ω is initialized to contain all operations without job predecessors.
2. Find $o^* = \arg\min_{o \in \Omega} f_o$, where f_o is the earliest completion time for operation o; $f_o = s_o + p_o$, where s_o is the earliest starting time for operation o, and p_o is the processing time for operation o.
3. a) Identify conflict set $C = \{o \in \Omega | m_o = m_{o^*} \land s_o < f_{o^*}\}$, where m_o is the machine associated with o.
   b) Select $o \in C$ according to the preference specified in the encoded representation.
   c) Add o to schedule S.
4. If schedule S is incomplete; remove o from Ω, add any immediate job successor of o to Ω, and continue from step 2.

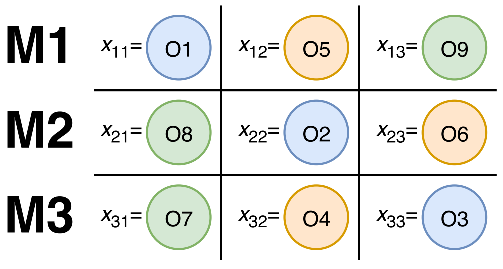

Figure 1: A JSSP schedule is represented as an $m \times n$ matrix of m machines and n jobs. Job 1 = $\langle O_1, O_2, O_3 \rangle$, Job 2 = $\langle O_4, O_5, O_6 \rangle$, Job 3 = $\langle O_7, O_8, O_9 \rangle$. Each row i encodes the ordering of operations for machine i; M_1, M_2, M_3.

PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) is an optimization technique mostly applied to problems with continuous solution spaces, where each particle in a swarm represents a solution. To solve a discrete problem such as JSSP, inspiration has been taken from [1]. The resulting algorithm uses a discrete, preference-based representation, a modified approach velocity update, a modified approach to position updates, a diversification strategy, and a local search. The overall algorithm can be described by the following three steps:

1. The initial swarm population is generated, where the position of each particle k is represented by a matrix of preferred machine operations x^k as shown in Figure 1. The job ordering for each machine is shuffled randomly, before the position of each particle x^k is developed by the G&T algorithm into a feasible schedule S^k.
2. For each solution S^k represented by particle k in the swarm, its best historical schedule, as determined by the schedule makespan, is tracked as *pbest*^{k}. The best solution out of all particles is tracked as *gbest*.
3. If the maximum number of iterations has not been reached:
   a) Update the velocity v^k for each particle k.
   b) For each particle k in the swarm; update its position x^k and develop the position x^k into the schedule S^k. For each schedule S^k, apply Taboo Search (TS), before updating *pbest*^{k} and *gbest* accordingly.

The velocity of each particle k is represented by an $m \times n$ matrix, where the velocity of each operation o_{ij} is denoted by $v_{ij} \in \{0, 1\}$, and o_{ij} is the operation of job j on machine i. When $v_{ij}^k = 1$, the operation o_{ij} of particle k has just been moved to its current position in x^k, and cannot be moved. In this way, the adapted velocity mechanism imitates the behavior of continuous velocity by providing the discrete representation with momentum. The velocity v_{ij}^k is randomly set to zero with a probability $(1 - w)$ when updating the velocity of each particle.

Parameter	Value
Iterations	150
Swarm size	100
Particle best factor (c_1)	0.5
Global best factor (c_2)	0.3
Probability of not resetting velocity (w)	0.5

Table 1: Particle Swarm Optimization (PSO) parameters.

Position updates relate to the discrete velocities. If $v_{ij}^k = 0$, the operation o_{ij}^k will be moved to the corresponding location of operation o_{ij}^k in $pbest^k$ with probability c_1, or it will be moved to the corresponding location of o_{ij}^k in *gbest* with probability c_2. In this manner, the *pbest* and *gbest* solutions are used to guide the global PSO search. The position update is performed as follows for each machine i:

1. Randomly select a location l in x_i^k.
2. Denote the operation at location l in x_i^k by O_1.
3. Find the location of operation O_1 in $pbest_i^k$ with probability c_1, or find the location of operation O_1 in $gbest_i$ with probability c_2. Denote the location found in $pbest_i^k$ or $gbest_i$ by l', and denote the operation at location l' in x_i^k by O_2.
4. If O_2 has been denoted, and the velocity components $v_{iJ_1}^k$ and $v_{iJ_2}^k$ for jobs J_1 and J_2, associated with operations O_1 and O_2, are both zero, then swap operations O_1 and O_2 in x_i^k, and set $v_{iJ_1}^k \leftarrow 1$.
5. If all locations in x_i^k has been considered, then the velocity update is completed. Otherwise, if $l < n$, set $l \leftarrow l+1$, else assign $l \leftarrow 1$, and continue from step 2.

To prevent the PSO search from getting stuck in local minima, a diversification strategy is used when updating $pbest^k$ and *gbest*. For each new solution S^k, the following update is used:

1. If the makespan of S^k is less than the makespan of *gbest*, set the *pbest* solution with the worst makespan to the current *gbest*, and set *gbest* to S^k.
2. Otherwise, if the makespan of S^k is equal to the makespan of any *pbest* or the *gbest* solution, replace the solution with the equal makespan with S^k.
3. Otherwise, if the makespan of S^k is better than the makespan of the worst *pbest* solution, set the worst *pbest* solution to S^k.

The implemented PSO uses Taboo search as described in [2] with backtracking and cycle detection. This local search, using parameters shown in Table 4, assists the global PSO search, which uses parameters shown in Table 1. The mutation operation described in [1] is not used.

Ant Colony Optimization

Ant Colony Optimization (ACO) is a swarm search inspired by how ants are able to locate food and direct other ants towards discovered food sources by using pheromones. The pheromone information is a central component in ACO by guiding the search through learning the relations between related operations as described in [3]. An operation o_a is related to an operation o_b if they are directly adjacent in a job sequence, or if they have to be processed by the same machine. The set of operations related to operation o_a is denoted by $\mathscr{R}_a$.

The pheromone information is represented as an $|O| \times |O|$ matrix $\mathscr{T}$, where an entry $\mathscr{T}_{ab}$ represents the desirability of processing operation o_a before operation o_b with a specific value denoted by τ_{ab}. Solutions are generated from the pheromone information using an algorithm similar to G&T, with the exception of how the conflict set C is chosen in step 3a, and how an operation is selected from C in step 3b. This process will always generated feasible schedules.

In 50% of cases, the conflict set is set to produce non-delay schedules; $C = \{o \in \Omega | s_o = \min_{o' \in \Omega} s_{o'}\}$. In the other cases, the conflict set is not limiting; $C = \Omega$.

The selection of an operation from the conflict set C is done by drawing randomly according to the probabilities:

$$p(o_a|\mathscr{T}) = \frac{\min_{o_b \in \mathscr{R}_a \cap O^+} \tau_{ab} \cdot \eta(o_a)^\beta}{\sum_{o_c \in C} \min_{o_b \in \mathscr{R}_a \cap O^+} \tau_{ab} \cdot \eta(o_a)^\beta}, \quad \forall o_a \in C \tag{1}$$

Where O^+ is the set of operations which has not yet been added to the schedule, and $\eta(o_a)$ is the heuristic information given by:

$$\eta(o_a) = \frac{\frac{1}{s_{o_a}+1}}{\sum_{o_c \in C} \frac{1}{s_{o_a}+1}} \tag{2}$$

The ACO search can be described by the following four steps, which are repeated for as long as necessary:

1. Generate $n_a = \max\left\{10, \lfloor \frac{|O|}{10} \rfloor\right\}$ solutions to form the generation $\mathscr{G}$, using the G&T algorithm with modified conflict set (Equation 1) and selection (Equation 2), where n_a is the number of ants.
2. Apply local search to improve each schedule in $\mathscr{G}$.
3. Apply TS to the schedule with the lowest makespan in $\mathscr{G}$.
4. Update the pheromones according to Equation 3 and 4, where η is the rate of pheromone evaporation, and S^* is the best schedule found by the ACO search.

$$\tau_{ab} \leftarrow \tau_{ab} + \rho \cdot [\delta(o_a, o_b) - \tau_{ab}] \tag{3}$$

$$\delta(o_a, o_b) = \begin{cases} 1, \text{if } o_a \text{ is processed before } o_b \text{ in } S^* \\ 0, \text{otherwise} \end{cases} \tag{4}$$

Pheromone values are limited to the range [0.001, 0.999].

Parameter	Value
Iterations	20
Heuristic power (β)	10.0
Evaporation rate (ρ)	0.1
Initial pheromone value (τ_{initial})	0.5

Table 2: Ant Colony Optimization (ACO) parameters.

Bees Algorithm

Bees Algorithm (BA) is a swarm search inspired by how bees locate and coordinate the gathering of food [4]. The implemented version of the algorithm for the JSSP problem can be described in 2 steps:

1. A number n of preference matrices are generated as shown in Figure 1, where the job ordering for each machine is shuffled, before each preference matrix is developed by the G&T algorithm into a schedule description. These n solutions represent the sites located by the n scouting bees.
2. While the search is not complete:

a) For each of the best e sites out of n, as determined by makespan, deploy nep bees. These bees each represent a TS search from the elite site. For each of the elite sites, the best solution out of nep solutions is used as a site in the next generation.

b) For each of the following best $(m-e)$ sites out of n, as determined by makespan, deploy nsp bees. These bees each represent a local search from the normal site. For each of the normal sites, the best solution out of nsp is used as a site in the next generation.

c) Any remaining $(n-m)$ scouting bees are used to locate new solutions which are generated in the same way as the initial sites in step 1 and used as sites in the next generation.

The original version of the algorithm [4] is used on continuous optimization problems, and the searching behavior for the elite and normal sites are the same, with the exception that more bees are used to search elite sites compared to normal sites. In the adapted version of the algorithm for the JSSP, a stronger local search is instead used to strengthen the search for elite sites, by employing a full TS for elite sites, and a simpler local search for normal sites.

Parameter	Value
Iterations	5
Number of scouts (n)	10
Number of sites selected (m)	8
Number of elite sites (e)	3
Number of normal sites ($m-e$)	5
Number of bees per elite site (nep)	1
Number of bees per normal site (nsp)	2

Table 3: Bees Algorithm (BA) parameters.

Taboo Search

Several papers [1][3] on the JSSP reference the TS detailed in [2]. This search is based on searching the neighborhood $H(S)$ of a schedule, which is established by the set of all moves deemed "interchanges near the borderline of blocks on a single critical path". Given a schedule, any longest sequence of operations in the schedule is called a critical path. An arbitrary selected critical path of S is considered. For this path, blocks of operations $B_1, \dots, B_r$ which share a machine is considered. The set of considered swapping moves $V(S)$ is defined as all moves which swap the last two operations in the first block, the first or last two operations in any middle block, or the first two operations in the last block. The neighborhood of S is then $H(S) = \{Q(S,v) : v \in V(S)\}$, where $Q(S,v)$ is the schedule resulting from applying move v to S.

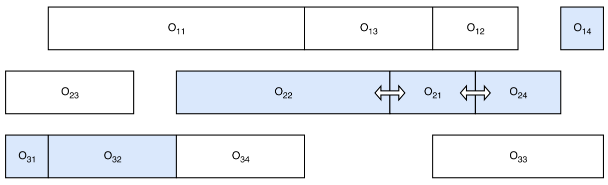

Figure 2: An example showing the critical path in shaded blue, and the set of possible swaps defined by $V(S)$.

Central to the TS is the Neighborhood Searching Procedure (NSP) which takes a schedule S, a set of moves V, a taboo list T of forbidden moves, and the best makespan found; C^*. The taboo list is a queue with maximum size $maxt$. The procedure returns a selected v' and the schedule S' which is the result of applying v' to S. It can be described in three steps:

1. Find the set of moves A which are forbidden by the taboo list, but which are "profitable", i.e. the moves which lead to a schedule S' with a lower makespan than C^*:

$$A = \{v \in V(S) \cap T : C_{\max}(Q(S,v)) < C^*\}, \tag{5}$$

where $C_{\max}(S)$ denotes the makespan of schedule S. If $(V(S) \setminus T) \cup A \neq \varnothing$, i.e. if there are any unforbidden moves or any profitable forbidden moves, then select a move v' such that:

$$C_{\max}(Q(S,v')) = \min\{C_{\max}(Q(S,v)) : v \in (V(S) \setminus T) \cup A\}, \tag{6}$$

and continue from step 3.

2. If $|V(S)| = 1$, then select $v' \in V(S)$. Otherwise, append the last move in the taboo list to the taboo list until $V(S) \setminus T \neq \varnothing$. Then select $v' \in V(S) \setminus T$.

3. Set $S' \leftarrow Q(S,v')$ and append v' to T.

In the case of the PSO and BA implementations, the local search used is to simply apply a single randomly selected move v' from the neighborhood $H(S)$ of a schedule, without a taboo list. In addition to using the same local search in the case of ACO, a full TS is also used to improve the best solution in each iteration. The Taboo Search consists of applying the Neighborhood Searching Procedure repeatedly to a schedule for a number of iterations, while keeping track of the best schedule and makespan found. Backtracking and cycle detection was also implemented as part of the TS as described in [2].

Parameter	Value
Iteration limit ($maxiter$)	150
Total iteration limit ($maxiter_{total}$)	1000
Taboo list limit ($maxt$)	5
Backtracking limit ($maxl$)	8
Max cycle detection count ($\max c$)	2
Max cycle detection duration ($\max\delta$)	100

Table 4: Taboo Search (TS) parameters.

References

[1] DY Sha and Cheng-Yu Hsu. A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering*, 51(4):791–808, 2006.

[2] Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management science*, 42(6):797–813, 1996.

[3] Christian Blum and Michael Sampels. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3(3):285–308, 2004.

[4] DT Pham, A Ghanbarzadeh, E Koc, S Otri, S Rahim, and M Zaidi. The bees algorithm-a novel tool for complex optimisation. In *Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference (3-14 July 2006)*. sn, 2011.

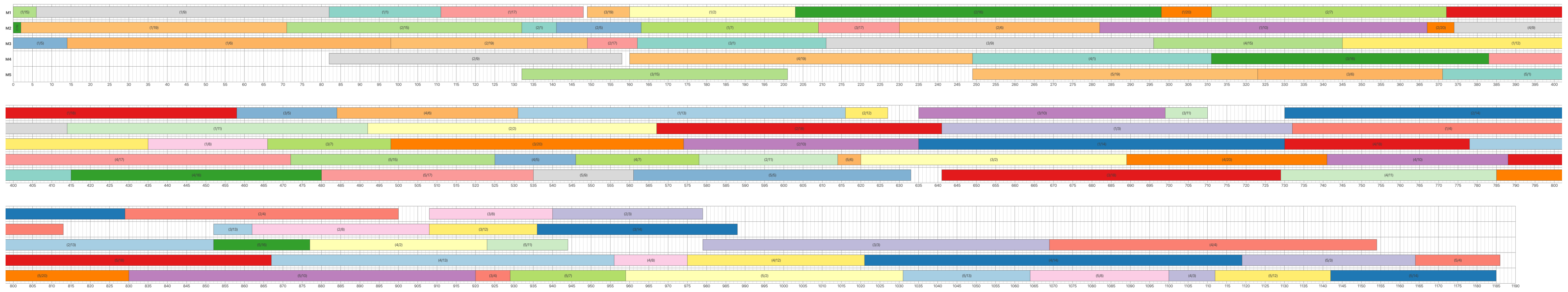

Figure 1: Particle Swarm Optimization (PSO) solution with makespan 1186 for problem 3 using parameters specified in Table 1 and 4.

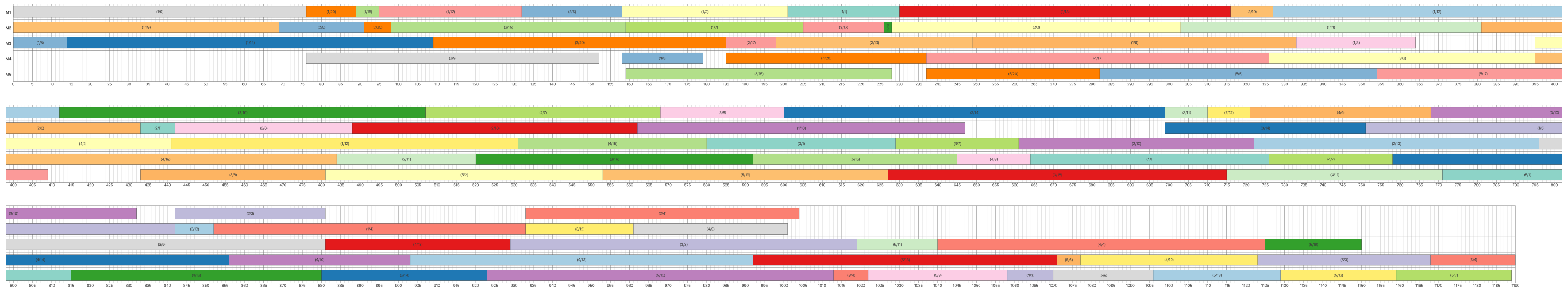

Figure 2: Ant Colony Optimization (ACO) solution with makespan 1190 for problem 3 using parameters specified in Table 2 and 4.

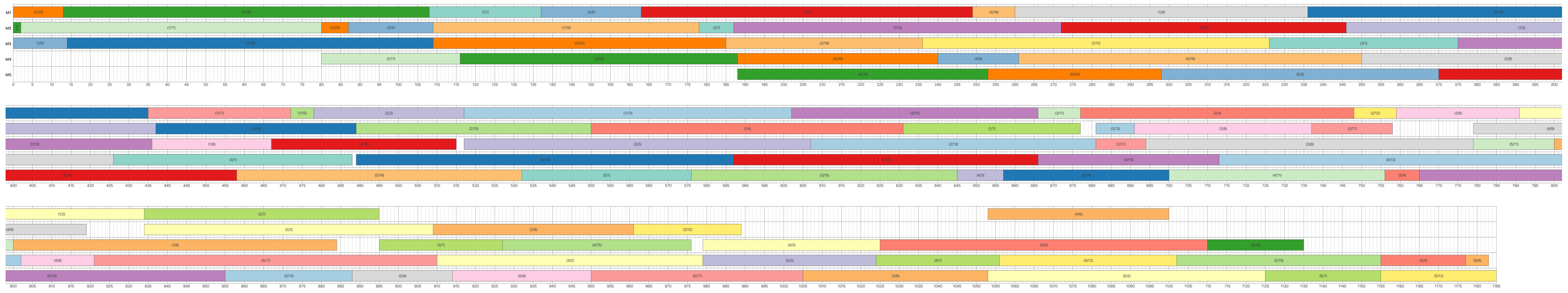

Figure 3: Bees Algorithm (BA) solution with makespan 1185 for problem 3 using parameters specified in Table 3 and 4.