Survey of One Machine Scheduling Problems

Kristina Šorić
Faculty of Economics Zagreb
Kennedyev trg 6
10000 Zagreb, Croatia
e-mail: ksoric@efzg.hr

May 6, 2002

Abstract

Below is presented a motivating introduction to One Machine Scheduling Problems is presented including a sample of their many applications and their many formulations as mixed integer programming problems. In the particular setting of these problems some new results related to Lot Sizing Problem with Equipment Replacement are reviewed. A new multicriteria mixed 0-1 integer programming problem is defined and the properties of efficient point set for this new problem are studied. Also, a heuristic and some new computational results are presented.

Key words: Scheduling, mixed integer programming, heuristic, metaheuristic, grouping problem, lot sizing, equipment replacement.

1 Scheduling Problems: Definition

In the last few years Machine Scheduling Problems became very actual and important problems in the field of Operational Research. A question that has received considerable attention is how to develop a production scheduling for a facility that processes one product at a time and incurs a changeover cost whenever it switches from the processing of one product to another. Even though this scheduling problem is a simplified representation of the actual planning problem, it has become a prototypical model in the operations management literature because it captures the major cost tradeoffs to be made in developing a schedule in numerous contexts: suppose that we have to perform a number of jobs by using a number of machines. To perform the jobs, each of them must be processed in the order given by a sequence. The processing of a job requires processing time. Each machine can process only one job at a time. Given a cost function by which the cost of each possible solution can be measured, we want to find a processing order on each machine so that the corresponding cost is minimized.

Such problems occur under widely varying circumstances. The terminology used above already suggests that the problem arose originally in an industrial production context. However, various other interpretations are possible: jobs and machines can stand for patients and hospital equipment, classes and teachers, ships and dockyards, dinners and cooks, programs and computers, cities and traveling salesman, projects and payments.

In order to formulate these problems let n be the number of jobs denoted by J_i $(i = 1 \dots n)$, m the number of machines and p_{ij} the processing time of the ith job on jth machine. Each machine can process only one job at a time.

Given a *cost function*, we want to find such a processing order on each machine so that the corresponding cost is minimized.

Also, it turns out to be useful to distinguish between *flow-shop* and *job-shop* problems. In the former case each job passes the machines in the same order, whereas in the latter case the machine order may vary per job.

Also, let us notice the difference between sequencing and scheduling. A feasible *sequence* simply corresponds to an ordering of the jobs on each machine. Corresponding to each feasible sequence there is an infinity of feasible *schedules* obtained by further specifying the exact starting time and finishing time of each job. In particular a schedule determines for each job a completion time $(C_i, i = 1 \dots, n)$ at which it is finished. Hence, every feasible schedule determines one sequence in a unique way . On the contrary, for every (feasible) sequence we can get the start times of every job summing the processing times of jobs processed before it.

For application of scheduling theory to be possible it is usually necessary that certain conditions are fulfilled. For example, a job J_i becomes available in a certain moment r_i $(i = 1 \dots n)$, so called, *release date*.

The second situation occurs when *precedence constraints* exist between jobs (for example, job J_i must be processed before job J_l, $i \neq l$) or the weights are assigned to them (where ω_i is the weight assigned to job J_i).

Also, there are situations when *deadlines* of jobs $(d_i, i = 1 \ldots n)$ have to be incorporated in the model (that is, $C_i \leq d_i, i = 1 \ldots n$), where C_i is the *completion* time of job $J_i, i = 1 \ldots n$).

If jobs may be rejected or left unfinished, the problem becomes more complicated. When the interruption of the jobs is allowed, we have so called *preemption*, otherwise a *nonpreemption* is present.

Each machine switch from one job to another requires so called *setup* time (where δ_i is the setup time of the job J_i) and *changeover* time (where h_i is the changeover time of the job J_i). Setup and changeover times may be *sequence independent* or *sequence dependent* (where δ_{ij} is the set up time of the job J_j precedeed by the job J_i and h_{ij} is the changeover time of the job J_j precedeed by the job J_i).

The description of the problem can be concluded with the discussion of the *optimality criteria*.

In order to discuss the objectives, let us assume the existence of cost functions $c_i(t)$ $(i = 1 \ldots n)$, non-decreasing in time variable. There are two types of criteria: we shall seek to minimize the maximum cost

$$\min\ c_{max} = \min(\max_i \{c_i(C_i)\}) \tag{1}$$

or minimize the total cost

$$\min \sum c_i = \min \sum_{i=1}^{n} c_i(C_i) \tag{2}$$

It is often advantageous from a computational point of view to use linear cost functions.

In the case that $c_i(t) = t$, $i = 1 \ldots n$, (1) corresponds to the *maximum completion time*, that is

$$C_{max} = \max_{i=1 \ldots n} C_i$$

and (2) corresponds to the *sum of completion times*, that is

$$\sum C_i = \sum_{i=1}^{n} C_i$$

If the cost function is of the form $c_i(t) = \omega_i t$ (where the ω_i are costs per time unit for every job J_i, $(i = 1 \dots n)$), (2) becomes the *weighted sum of completion times*, that is

$$\sum \omega_i C_i = \sum_{i=1}^{n} \omega_i C_i \tag{3}$$

If $\omega_i = \frac{1}{n}$, $i = 1 \dots n$, (3) becomes the *average completion times*, that is

$$\bar{C} = \frac{1}{n} \sum_{i=1}^{n} C_i$$

If we define the *flow time*

$$F_i := C_i - r_i, \;\; i = 1 \dots n$$

we can minimize the *sum of the flow times* $\sum_{i=1}^{n} F_i$, the *weighted sum of flow times* $\sum_{i=1}^{n} \omega_i F_i$ and the *average flow* time $\bar{F}$.

Finally, we define the *waiting time* W_i by

$$W_i := C_i - (r_i + \sum_{j=1}^{m} p_{ij}), \; i = 1 \dots n$$

Putting $c_i(t) = t - (r_i + \sum_{j=1}^{m} p_{ij})$, we obtain criteria corresponding to the *sum of waiting times* $\sum W_i$, the *weighted sum of waiting times* $\sum \omega_i W_i$ and the *average waiting time* $\bar{W}$.

As pointed before, it is sometimes convenient to assume the deadlines $(d_i, \; i = 1 \dots n)$ incorporated in the model. Taking $c_i(t) = t - d_i$, we define the *lateness* L_i by

$$L_i := C_i - d_i, \; i = 1 \dots n$$

and arrive at criteria corresponding to the minimization of the *maximum lateness* L_{max}, the *sum of lateness* $\sum L_i$, the *weighted sum of lateness* $\sum \omega_i L_i$ and the *average lateness* $\bar{L}$.

Further, we define the *tardiness* by

$$T_i := \max\{0, L_i\},\ i = 1 \ldots n$$

and we may seek the minimization of the *maximum tardiness* T_{max}, the *sum of tardiness* $\sum T_i$, the *weighted sum of tardiness* $\sum \omega_i T_i$ or the *average tardiness* $\bar{T}$.

Finally, there are criteria based on inventory cost, criteria based on utilization and the others.

We can observe that some criteria are equivalent in the sense that the optimal solution with respect to one of the criteria is optimal with respect to the other(s) ([3]).

Now, we can give the classification (according to [3])of the scheduling problems which has the following format:

$$\langle \alpha \mid \beta \mid \gamma, \Gamma \mid \Delta \rangle$$

where:

- α represents the number of jobs
- β represents the number of machines
- γ indicates the possible restriction (flow-shop o job-shop, sequence dependent or sequence independent)
- Γ indicates the possible limitations on p_{ij}, d_i o ω_i
- Δ indicates the optimality criterion

For example, the problem

$$\langle n \mid 1 \mid seq\ dep \mid C_{max} \rangle$$

is the problem to minimize the maximum completion time of n jobs to be processed on one machine, when there are job dependent setup times and sequence dependent changeover times.

Observe that the above classification does not describe the problem completely because a change in some parameter often leads to a substantially different problem that is the subject of separate research.

2 One Machine Scheduling Problems

A special class of scheduling problems arises from production planning. Its objective is to find an optimal schedule of jobs that allows production runs for several jobs that are made on a single machine in such a way that setup and inventory costs or production completion time or average workload are minimized.

Example 1. The simplest formulation of one machine scheduling problem is the following: there are n jobs. Let p_i be the processing time required by the ith job on the machine. Assume that the machine can only process one job at a time. Also, when a job is put on the machine, its processing must be completed before the machine can take up another job. Let t_i represent the starting time of processing the ith job, $i = 1, \dots, n$. The problem is to determine the constraints on the time variable t_i's so that they represent a feasible sequence on the machine.

This problem is modelled as a mixed integer programming problem in the following way: let

$$y_{ij} = \begin{cases} 1, & \text{if job } J_i \text{ is started on the machine before job } J_j \\ 0, & \text{otherwise} \end{cases}$$

$$i, j = 1, \dots, n, \ i \neq j$$

Let α be a very large positive number. The constraints are:

$$\alpha y_{ij} + t_i - t_j \geq p_j$$

$$\alpha(1 - y_{ij}) + t_j - t_i \geq p_i$$

$$y_{ij} + y_{ji} = 1$$

$$y_{ij} \in \{0, 1\}, t_i \geq 0, \ i, j = 1 \dots, n$$

Example 2. If the switch from one job to ith job requires setup time δ_i the constraints are the following: $\alpha y_{ij} + t_i - t_j \geq \delta_i + p_j$, $\alpha(1 - y_{ij}) + t_j - t_i \geq \delta_j + p_i$, $y_{ij} + y_{ji} = 1$, $y_{ij} \in \{0, 1\}, t_i \geq 0,\ i, j = 1 \ldots, n$.

This is the example of a sequence independent one machine scheduling problem.

Example 3. If the switch from ith job to jth job requires setup time δ_{ij}, we have the sequence dependent one machine scheduling problem. Its mixed integer programming formulation is then much more complicated than the above one.

$$x_i^k = \begin{cases} 1, & \text{if job } J_i \text{ is } k\text{th in the sequence} \\ 0, & \text{otherwise} \end{cases}$$

$$i, k = 1, \ldots, n$$

$$w_{ij}^k = \begin{cases} 1, & \text{if job } J_i \text{ is } k-1\text{th and job} J_j \text{ is } k\text{th in the sequence} \\ 0, & \text{otherwise} \end{cases}$$

$$k = 2, \ldots, n,\ i \neq j$$

Let α be a very large positive number. The constraints are:

$$\alpha y_{ij} + t_i - t_j \geq \sum_{k=2}^{n} w_{ji}^k \delta_{ji} + p_j \tag{4}$$

$$\alpha(1 - y_{ij}) + t_j - t_i \geq \sum_{k=2}^{n} w_{ij}^k \delta_{ij} + p_i \tag{5}$$

$$y_{ij} + y_{ji} = 1 \tag{6}$$

$$\sum_{i=1}^{n} x_i^k = 1 \quad k = 1, \ldots, n \tag{7}$$

$$\sum_{k=1}^{n} x_i^k = 1 \quad i = 1, \ldots, n \tag{8}$$

$$x_i^k + \sum_{r=1}^{n-k} x_j^{k+r} - 1 \le y_{ij} \quad i \ne j,\ k = 1, \ldots, n \tag{9}$$

$$w_{ij}^k \ge x_i^{k-1} + x_j^k - 1 \quad i \ne j,\ k = 2, \ldots, n \tag{10}$$

$$w_{ij}^k \le x_i^{k-1} \quad i \ne j,\ k = 2, \ldots, n \tag{11}$$

$$w_{ij}^k \le x_j^k \quad i \ne j, \quad k = 2, \ldots, n \tag{12}$$

$$w_{ij}^k + w_{ji}^k \le 1 \quad i \ne j, \quad k = 2, \ldots, n \tag{13}$$

$$\sum_{k=2}^{n} w_{ij}^k \le 1 \quad i \ne j \tag{14}$$

$$\sum_{j=1, j \ne i}^{n} w_{ij}^k \le 1 \quad i = 1, \ldots, n, \quad k = 2, \ldots, n \tag{15}$$

$$\sum_{i=1, i \ne j}^{n} w_{ij}^k \le 1 \quad j = 1, \ldots, n, \quad k = 2, \ldots, n \tag{16}$$

$$y_{ij}, x_i^k, w_{ij}^k \in \{0, 1\}, t_i \ge 0$$

The conditions (4) and (5) describe the fact that if the job J_i directly precedes the job J_j on the kth place, then it should be $t_j - t_i \ge \delta_{ij} + p_i$. (7) and (8) make sure that the job J_i can be only on the one place in the sequence and vice versa. (9) says that if the job J_j precedes the job J_i than $x_j^{k+r} = 0,\ \forall\, r = 1, \ldots, n-k$. (10) implies that if $x_i^{k-1} = 1, x_j^k = 1 \Rightarrow w_{ij}^k = 1$, (11) implies that if $x_i^{k-1} = 0 \Rightarrow w_{ij}^k = 0$, (12) implies that if $x_j^k = 0 \Rightarrow w_{ij}^k = 0$ and (13) implies that if $w_{ij}^k = 1 \Rightarrow w_{ji}^k = 0$.

Some other constraints could be added to the problem like release data

$$t_i \ge r_i,$$

deadlines

$$t_i + p_i \leq d_i,$$

precedence constraints and so on.

Remark For Example 1., we could add some valid inequalities:

$$t_j \geq \sum_{i \neq j} p_i y_{ij}, \quad j = 1, \ldots, n$$

$$\alpha(1 - y_{ij}) + t_j - t_i \geq p_i + \sum_{l \neq i,j} p_l (y_{lj} - y_{li}), \quad i, j = 1, \ldots, n$$

or, for Example 2., we have

$$t_j \geq \sum_{i \neq j} (p_i + \delta_i) y_{ij} + \delta_j, \quad j = 1, \ldots, n$$

$$\alpha(1 - y_{ij}) + t_j - t_i \geq p_i + \sum_{l \neq i,j} (p_l + \delta_l)(y_{lj} - y_{li}) + \delta_j, \quad i, j = 1, \ldots, n$$

Also, even if we could consider stochastic scheduling problems too, we will concentrate on deterministic ones. More about scheduling problems could be found on http://www.nada.kth.se/ viggo/wwwcompendium/node173.html.

Example 4. In order to avoid a big real number α, we give another formulation of sequence dependent One Machine Scheduling Problem.

$$x_i^k = \begin{cases} 1, & \text{if job } J_i \text{ is } k\text{th in the sequence} \\ 0, & \text{otherwise} \end{cases}$$

$$i, k = 1, \ldots, n$$

$$z_{lij}^k = \begin{cases} 1, & \text{if job } J_i \text{ is } k-1\text{th and job} J_j \text{ is } k\text{th in the sequence} \\ & \text{and both of them precede the job} J_j \\ 0, & \text{otherwise} \end{cases}$$

$$k = 2, \ldots, n-1,\ i \neq l \neq j$$

It follows

$$t_j \geq \sum_{k=2}^{n-1} \sum_{l \neq i \neq j} \delta_{li} z_{lij}^k + \sum_{i \neq j} p_i y_{ij}, \quad j = 1, \ldots, n \tag{17}$$

$$\sum_{i=1}^{n} x_i^k = 1 \quad k = 1, \ldots, n \tag{18}$$

$$\sum_{k=1}^{n} x_i^k = 1 \quad i = 1, \ldots, n \tag{19}$$

$$x_i^k + \sum_{r=1}^{n-k} x_j^{k+r} - 1 \leq y_{ij} \quad i \neq j,\ k = 1, \ldots, n \tag{20}$$

$$x_i^k - \sum_{r=1}^{k} x_j^r \leq \sum_{l=1}^{n} z_{lij}^k \quad i \neq j,\ k = 2, \ldots, n-1 \tag{21}$$

$$x_l^{k-1} - \sum_{r=1}^{k} x_j^r \leq \sum_{i=1}^{n} z_{lij}^k \quad l \neq j,\ k = 2, \ldots, n-1 \tag{22}$$

$$\sum_{l=1}^{n} \sum_{i \neq l} z_{lij}^k \leq 1 - x_j^k \quad j = 1, \ldots, n,\ k = 2, \ldots, n-1 \tag{23}$$

$$\sum_{l=1}^{n} \sum_{i \neq l} z_{lij}^k \leq 1 - x_j^{k-1} \quad j = 1, \ldots, n,\ k = 2, \ldots, n-1 \tag{24}$$

$$z_{lij}^k \leq x_l^{k-1} \quad l \neq i \neq j,\ k \tag{25}$$

$$z_{lij}^k \leq x_i^k \quad l \neq i \neq j,\ k \tag{26}$$

$$z_{lij}^k \leq y_{ij} \quad l \neq i \neq j,\ k \tag{27}$$

$$z_{lij}^k \leq y_{lj} \quad l \neq i \neq j,\ k \tag{28}$$

$$z_{lij}^k \geq x_l^{k-1} + x_i^k + y_{lj} + y_{ij} - 3 \quad l \neq i \neq j,\ k \tag{29}$$

$$y_{ij}, x_i^k, z_{lij}^k \in \{0,1\}, t_i \geq 0$$

The conditions (17) count all the setup times and processing times occurred before the moment t_j. To explain the conditions (21) let job J_l be $k-1$th in the sequence. If $\sum_{l=1}^n z_{lij}^k = 0$ than $z_{lij}^k = 0$, $\forall i \neq j$. That means (a) J_i is not kth in the sequence, that is $x_i^k = 0$ and than $-\sum_{r=1}^k x_j^r \leq 0$ or (b) J_i is kth in the sequence, but not before J_j. Since $x_i^k = 1$ from conditions (21) and (19) follows that there is some $r < k$ such that $x_j^r = 1$, that is job $J-j$ is precessed before job J_i. Conditions (22) are explained in the analogous way as the conditions (21). The conditions (23) describe the fact that if $x_j^k = 1$, that is, the job J_j is kth in the sequence, then $z_{lij}^k = 0$, $\forall l \neq i \neq j$. If $x_j^k = 0$, the conditions are satisfied in a trivial way. (25) say that if the job J_l is not $k-1$th in the sequence, then $z_{lij}^k = 0$. Analogously, (26), if the job J_i is not kth in the sequence, then $z_{lij}^k = 0$. The conditions (27) make sure that if the job J_i is not processed before J_j, then $z_{lij}^k = 0$. Analogously, (28) make sure that if the job J_l is not processed before J_j, then $z_{lij}^k = 0$. Finally, the conditions (29) describe the fact that if $x_l^{k-1} = 1$, $x_i^k = 1$, $y_{lj} = 1$ and $y_{ij} = 1$ then $z_{lij}^k = 1$. Every other case is satisfied in a trivial way.

Example 5. Time Indexed Formulation. The very important question in Machine Scheduling Problems is how to model the time. A well known alternative way to model it is to use time-indexed variables. Such formulations are often unavoidable when manpower or resource requirements vary during the processing of a job. To present some basic results for such formulation, suppose $p_i, r_i, t_i, C_i \in N$. We define the problem of minimizing the weighted sum of completion times $\sum_{i=1}^n \omega_i C_i$ with release data r_i. Let T be the time horizon and

$$y_{it} = \begin{cases} 1, & \text{if job } J_i \text{ is finished in period } t \\ 0, & \text{otherwise} \end{cases} \quad i = 1, \ldots, n, \quad t = 1, \ldots, T$$

Since the job J_i can not start before its release data and finish before $r_i + p_i$, it follows that

$$y_{it} = 0, \quad \forall t < r_i + p_i$$

.

Also, let us add the condition that job J_i has to precede the job J_k. The model is as follows:

$$\min \sum_{i=1}^{n} \omega_i \sum_{t=1}^{T} t y_{it}$$

$$\sum_{t=1}^{T} t y_{it} = 1, \quad i = 1, \ldots, n \tag{30}$$

$$\sum_{s=1}^{t} y_{is} \geq \sum_{s=1}^{t+p_k} y_{ks} \tag{31}$$

$$\text{za} J_i < J_k, \quad t = 1, \ldots, T - p_k$$

$$\sum_{i=1}^{n} \sum_{s=t}^{t+p_i-1} y_{is} \leq 1, \quad t = 1, \ldots, T \tag{32}$$

The condition (30) describes the fact that each job can be performed once at the most. (31) makes sure that job J_i precedes the job J_k. Since there can be only one job on the machine, we have (32). If we want job J_i to be preformed at least τ periods before job J_k, we will put

$$\sum_{t=1}^{T} t y_{kt} - \sum_{t=1}^{T} t y_{it} \geq \tau$$

3 Resolving One Machine Scheduling Problems

(One) Machine Scheduling Problems are modelled as mixed integer programming problems that are usually solved by branch and bound method.

3.1 Branch and Bound

This method is almost as frequently applied to (mixed) integer programming problems as the simplex method to linear programming problems. In a situation where it takes a long time to reach an optimal solution, the calculations can be stopped at any stage of the branch and bound process and the best solution obtained so far can be used. However, similar to the simplex method, the worst case complexity of branch and bound algorithm is exponential.

Sometimes authors use cutting planes to reformulate and solve mixed integer programming problems using valid inequalities as cutting planes. The resulting algorithm is called branch and bound/cutting planes algorithm. One of this algorithms for One Machine Scheduling Problem could be found in [7].

Since the early 1970s, a class of optimization problems is being constructed for which the existence of efficient algorithms is extremely unlikely. An interesting feature of this class is the strong interrelationship between its members: if one finds an efficient algorithm to solve any one problem in this class, one can easily modify it to solve all the problems in this class efficiently. This class is usually referred to as the class of *NP-hard problems*. NP-hard problems include some (One) Machine Scheduling Problems.

The conjecture that no NP-hard problem is efficiently solvable is denoted by

$$P \neq NP$$

where P refers to the class of problems for which a polynomial algorithm is known. So, what remains in this particular area of research is to prove (or disprove) the $P \neq NP$ conjecture. Until someone disproves this conjecture, only algorithms for NP-hard problems are designed that generate good feasible solutions in polynomial time. Such algorithms are called *heuristics*. The current popularity of heuristics are classified, tested and mutually compared on their performances.

More about the complexity of (One) Machine Scheduling Problems can be found on http://www.mathematik.uni-osnabrueck.de/research/OR/class/ and http://www.nada.kth.se/ viggo/wwwcompendium/.

3.2 Heuristics

In mathematical programming, heuristics usually means a procedure that seeks a solution but does not guarantee it will find one. It is often used in contrast to an algorithm, so branch and bound would not be considered a heuristics in this sense. Heuristic search is any (purposeful) search procedure to seek a solution to a global optimization problem, notably to combinatorial optimization. One of this algorithms for One Machine Scheduling Problem could be found in [6].

A specific class of local heuristic search algorithms is the *greedy algorithm*.

Here are heuristic search strategies that are based on some biological metaphor:

- Ant colony optimization, based on how ants solve problems;
- Genetic algorithm, based on genetics and evolution;
- Neural networks, based on how the brain functions;
- Simulated annealing, based on thermodynamics;
- Tabu search, based on memory-response;
- Target analysis, based on learning.

Greedy algorithm applies when the optimization problem is to decide whether or not to include some element from a given set. A greedy algorithm begins with no elements and sequentially selects an element from the feasible set of remaining elements by myopic optimization. (The elements could have been sorted by some criterion, such as associated weights.) This results in an optimal solution to the problem if, and only if, there is an underlying matroid structure (for example, see spanning tree).

Ant colony optimization is a heuristic search approach to combinatorial optimization based on the behavior of ant colonies, particularly their ability to collectively determine the shortest paths . Real ants are capable of finding shortest path from a food source to the nest without using visual cues. Also, they are capable of adapting to changes in the environment, for example finding a new shortest path once the old one is no longer feasible due to a new obstacle.

Genetic algorithm (GA) is a class of algorithms inspired by the mechanisms of genetics, which has been applied to global optimization (especially combinatorial optimization problems). It requires the specification of three operations (each is typically probabilistic) on objects, called "strings" (these could be real-valued vectors):

- Reproduction - combining strings in the population to create a new string (offspring)
- Mutation - spontaneous alteration of characters in a string
- Crossover - combining strings to exchange values, creating new strings in their place

These can combine to form hybrid operators, and the reproduction and crossover operations can include competition within populations. Here is a generic GA (strategy):

1. Initialize population.
2. Select parents for reproduction and GA operators (viz., mutation and crossover).
3. Perform operations to generate intermediate population and evaluate their fitness values.
4. Select members of population to remain with new generation.

Repeat 1-3 until some stopping rule is reached.
More details about GA could be found on http://samizdat.mines.edu/ga_tutorial/.

Neural network (also called artificial neural network, abbr. ANN). A network where the nodes correspond to neurons and the arcs correspond to synaptic connections in the biological metaphor.

Simulated annealing is an algorithm for solving hard problems, notably combinatorial optimization, based on the metaphor of how annealing works:

reach a minimum energy state upon cooling a substance, but not too quickly in order to avoid reaching an undesirable final state. As a heuristic search, it allows a non-improving move to a neighbor with a probability that decreases over time. The rate of this decrease is determined by the cooling schedule, often just a parameter used in an exponential decay (in keeping with the thermodynamic metaphor). With some (mild) assumptions about the cooling schedule, this will converge in probability to a global optimum.

Tabu search is a metaheuristics to solve global optimization problems, notably combinatorial optimization, based on multi-level memory management and response exploration. It requires the concept of a neighborhood for a trial solution (perhaps partial). In its simplest form, a tabu search appears as follows:

- Initialize. Select x and set Tabu List T = null. If x is feasible, set $x^* = x$ and $f^* = f(x^*)$; otherwise, set $f^* = -\infty$ (for minimization set $f^* = \infty$).
- Select move. Let $S(x)$ = set of neighbors of x. If $S(x) \setminus T$ is empty, go to update. Otherwise, select y in $\arg\max\{E(v) : v \in S(x) \setminus T\}$, where E is an evaluator function that measures the merit of a point (need not be the original objective function, f). If y is feasible and $f(y) > f^*$, set $x^* = y$ and $f^* = f(x^*)$. Set $x = y$ (i.e., move to the new point).
- Update. If some stopping rule holds, stop. Otherwise, update T (by some tabu update rule) and return to select move.

There are many variations, such as aspiration levels, that can be included in more complex specifications.

Target analysis is a metaheuristics to solve global optimization problems, notably combinatorial optimization, using a learning mechanism. In particular, consider a branch and bound strategy with multiple criteria for branch selection. After solving training problems, hindsight is used to eliminate dead paths on the search tree by changing the weights on the criteria: set $w > 0$ such that $wV_i \leq 0$ at node i with value, V_i, that begins a dead path, and $wV_i > 0$ at each node, i, on the path to the solution. If such weights exist, they define a separating hyperplane for the test problems. If such weights do not exist, problems are partitioned into classes, using a form of feature analysis, such that each class has such weights for those test problems in the class. After training is complete, and a new problem arrives, it is first classified, then those weights are used in the branch selection.

Remark Metaheuristics is a general framework for heuristics in solving hard problems. The idea of 'meta' is that of level. An analogy is the use of a meta-language to explain a language.

Metaheuristics guide the application and use of local improvement heuristics. They are used to search the many local optima which most local heuristics find, attempting to locate the global optimum. This approach has met with a great deal of success.

Examples of metaheuristics are: Ant colony optimization, Genetic algorithms, Memetic algorithms, Neural networks, Scatter search, Simulated annealing, Tabu search, Target analysis.

Memetic Algorithms is a population-based approach for heuristic search in optimization problems. They have shown that they are orders of magnitude faster than traditional Genetic Algorithms for some problem domains. Basically, they combine local search heuristics with crossover operators. For this reason, some researchers have viewed them as Hybrid Genetic Algorithms. However, combinations with constructive heuristics or exact methods may also belong to this class of metaheuristics. Since they are most suitable for MIMD parallel computers and distributed computing systems (including heterogeneous systems) as those composed by networks of workstations, they have also received the dubious denomination of Parallel Genetic Algorithms. Other researchers know it as Genetic Local Search.

3.3 Defining One Machine Scheduling Problem as a Grouping Problem

Automatic grouping and segmentation of images remains a challenging problem in computer vision. Recently, a number of authors have demonstrated good performance on this task using methods that are based on eigenvectors of the affinity matrix. These approaches are extremely attractive in that they are based on simple eigendecomposition algorithms whose stability is well understood.

Human perceiving of a scene can often easily segment it into coherent segments or groups. There has been a tremendous amount of effort devoted to achieving the same level of performance in computer vision. In many cases, this is done by associating with each pixel a feature vector (e.g. color, motion, texture, position) and using a clustering or grouping algorithm on these feature vectors.

Recently, treating the grouping problem as a graph partitioning problem, a number of authors have suggested alternative segmentation methods that are based on eigenvectors of the (possibly normalized) "affinity matrix". One example of such an affinity matrix is defined by:

$$W(i,j) = e^{-d(x_i,x_j)/2\sigma^2}$$

with σ a free parameter. In this case we have used $d(x_i, x_j) = \|x_i - x_j\|$, but different definition of affinities are possible. The affinities do not even have to obey the metric axioms, we only assume that $d(x_i, x_j) = d(x_j, x_i)$.

From visual inspection, the affinity matrix contains information about the correct segmentation. Shi and Malik [4] look at generalized eigenvectors. Let D be the degree matrix of W:

$$D(i,i) = \sum_j W(i,j)$$

Define the generalized eigenvector y_i as a solution to:

$$(D - W)y_i = \lambda_i D y_i$$

and define the second generalized eigenvector as the y_i corresponding to the second smallest λ_i. Shi and Malik suggested thresholding this second generalized eigenvector of W in order to cut the image into two parts. They have shown that the second generalized eigenvector is a solution to a continuous version of a discrete problem in which the goal is to minimize:

$$\frac{y^T(D - W)y}{y^T D y}$$

subject to the constraint that $y \in \{1, -b\}$ and $y^T D 1 = 0$ (where 1 is the vector of all ones). Note that the above expression is the Rayleigh quotient ([1]).

The significance of the discrete problem is that its solution can be shown to give you the segmentation that minimizes the normalized cut:

$$Ncut(A,B) = \frac{cut(A,B)}{asso(A,V)} + \frac{cut(A,B)}{asso(B,V)}$$

where $cut(A,B) = \sum_{i \in A, j \in B} W(i,j)$ and $asso(A,V) = \sum_j \sum_{i \in A} W(i,j)$. Thus the solution to the discrete problem finds a segmentation that minimizes the affinity within each group.

The **Grouping Algorithm** consists of the following steps:

1. Given an image or image sequence, set up a weighted graph G=(V,E) and set the weight on the edge connecting two nodes being a measure of the similarity (affinity) between nodes.
2. Solve $(D-W)x = \lambda Dx$ for eigenvectors with the smallest eigenvalues.
3. Use the eigenvector with second smallest eigenvalue to bipartition the graph
4. Decide if the current partition should be sub-divided and recursively repartition the segmented parts if necessary

To define an One Machine Scheduling Problem as a grouping problem let jobs be the nodes of a graph, the sum of set up time and processing time be the weight on an edge. Since in a sequence dependent scheduling problem case, the affinity matrix is not a simetric matrix, we can do the following: between any two nodes i and j, put a node (i,j) such that the weight on the edge connecting nodes i and (i,j) is δ_{ij} and the weight on the edge connecting nodes (i,j) and j is δ_{ji}. After applying the grouping algorithm, we get a bipartition of the graph. Treating each group of the bipartition as a job, we resolve the scheduling problem with two jobs. The processing time of each job is the worst case sum of all set ups and processing times in that group of the bipartition. repeating the procedure, we stop when the difference between two consecutive optimal solutions is sufficiently small.

This definition of One Machine Scheduling Problem as a grouping problem is just an idea discussed in [8].

4 Lot Sizing Problem

Production Lot Sizing Problem is one of the oldest mixed-integer programs in operations research, first presented by Wagner and Whitin in 1958. It is defined as follows: each of m products (jobs) is to be processed on a single machine in order to satisfy known demands in each of n periods. The object is to minimize the sum of the costs of production, storage and set up. The data are:

- d_t^i the demand for product i in period t
- p_t^i the unit production cost of product i in period t
- h_t^i the unit storage cost for product i in period t
- k_t^i the unit backlog (shortage) cost for product i in period t
- f_t^i the fixed setup cost for product i in period t
- c_t^{ij} the changeover cost from product i to product j in period t
- C_t^i the production capacity for product i in period t

Let

- x_t^i the amount of product i produced in period t
- s_t^i the inventory (stock) of product i in period t
- r_t^i the backlog (shortage) of product i in period t

and

$$y_t^i = \begin{cases} 1, & \text{if machine is set up for product } i \text{ in period } t \\ 0, & \text{otherwise} \end{cases}$$

$$w_t^{ij} = \begin{cases} 1, & \text{if machine is set up for product } j \text{ in period } t \\ & \text{and was set up for product } i \text{ in period } t-1 \\ 0, & \text{otherwise} \end{cases}$$

The mixed integer programming formulation is

$$min \sum_{i=1}^{m} \sum_{t=1}^{n} (p_t^i x_t^i + h_t^i s_t^i + k_t^i r_t^i + f_t^i y_t^i + \sum_{j \neq i} c_t^{ij} w_t^{ij})$$

$$s_{t-1}^i + r_t^i + x_t^i = d_t^i + s_t^i + r_{t-1}^i \quad \forall i, t \tag{33}$$

$$x_t^i \leq M y_t^i \quad \forall i, t \tag{34}$$

$$w_t^{ij} \geq y_{t-1}^i + y_t^j - 1 \tag{35}$$

$$\sum_{i=1}^{m} y_t^i \leq 1, \quad \forall t \tag{36}$$

$$x_t^i, s_t^i, r_t^i \geq 0, \;\; y_t^i, w_t^{ij} \in \{o, 1\}$$

where $M$is the upper bound on the production capacities. The constraints (33) represent the flow conservation constraints for each product in each period. The constraints (36) refer to a single mode of production.

(34) could be changed by the condition

$$x_t^i \leq C_t^i y_t^i \quad \forall i, t$$

The fact that the production in each period is either 0 or at the full and constant capacity over time is modelled by the condition

$$x_t^i = C_t^i y_t^i \quad \forall i, t$$

5 Lot Sizing Problem with Equipment Replacement

An Equipment Replacement Problem concerns a machine which deteriorates with age and the decision to replace it. We assume that we must own such a machine during each of n periods and that the utility of operating a machine for one period is a known quantity and depends on the age of the machine. The object is to decide when to replace the machine in order to maximize the utility of operating it.

Combining the Lot Sizing Problem with this one, a new multicriteria mixed integer programming problem is defined as follows: let

- j the age of the machine, $j = 1 \dots J$
- π^i_{tj} unit production revenue of product $i$in the period $t$if the machine is of age j
- c^j_t cost of replacing the machine of age $j$in period t
- m^j_t cost of maintaining the machine of age $j$in period t

The variables are:

- x^i_{tj} the amount of product i produced in period $t$if the machine is of age j
- $$z^j_t = \begin{cases} 1, & \text{if machine of age } j \text{ is replaced in period } t \\ 0, & \text{otherwise} \end{cases}$$

with $\sum_{j=1}^{J} z^j_0 = 1$.

The multicriteria mixed integer programming formulation of Lot Sizing Problem with Equipment Replacement (LSPER) is the following one:

$$min \sum_{t=1}^{n} \left[\sum_{i=1}^{m} \left(p^i_t \sum_{j=1}^{J} x^i_{tj} + h^i_t s^i_t + f^i_t y^i_t \right) + \sum_{j=1}^{J} \left(c^j_t z^j_t + m^j_t (1 - z^j_t) \right) \right]$$

$$max \sum_{t=1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{J} \pi^i_{tj} x^i_{tj}$$

$$s^i_{t-1} + \sum_{j=1}^{J} x^i_{tj} = d^i_t + s^i_t \quad \forall i, t \tag{37}$$

$$\sum_{j=1}^{J} x_{tj}^i \leq M y_t^i \quad \forall i,t \tag{38}$$

$$x_{tj}^i \leq M \sum_{l=1}^{J} z_{t-j}^l \quad \forall i,t,j, \;\; j \leq t \tag{39}$$

$$\sum_{j=1}^{J} x_{tj}^i \leq M \left(1 - \sum_{j=1}^{J} z_t^j\right) \quad \forall i,t \tag{40}$$

$$\sum_{i=1}^{m} y_t^i \leq 1, \quad \forall t \tag{41}$$

$$\sum_{j=1}^{J} z_t^j \leq 1, \quad \forall t \tag{42}$$

$$\sum_{l=t+1}^{t+J} \sum_{j=1}^{l-t} z_l^j \geq \sum_{j=1}^{J} z_t^j, \quad \forall t = 1, \ldots n - J \tag{43}$$

$$x_{tj}^i, s_t^i \geq 0, \;\; y_t^i, z_t^j \in \{o, 1\}$$

Constraints (39) force the production of product i in period t to zero if the machine is not of age j. If the machine of age less or equal to $J$is replaced in period t, the production x_{tj}^i in this period is forced to zero by constraints (40). Constraints (42) represent the possibility of changing the machine only once in period t. Constraints (43) force the machine to be replaced after age J.

5.1 Resolving the *LSPER*

Let us begin with the construction of efficient point set as follows: let

$$f_1 = -\sum_{t=1}^{n} \left[\sum_{i=1}^{m} \left(p_t^i \sum_{j=1}^{J} x_{tj}^i + h_t^i s_t^i + f_t^i y_t^i \right) + \sum_{j=1}^{J} \left(c_t^j z_t^j + m_t^j (1 - z_t^j) \right) \right]$$

$$f_2 = \sum_{t=1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{J} \pi_{tj}^i x_{tj}^i$$

and S the feasible set of $LSPER$. We define the objective set as

$$F = F(S) = \{(f_1, f_2), f_1 = f_1(w), f_2 = f_2(w), w \in S\}.$$

The problem $LSPER$ is now stated as

$$max\{(f_1, f_2), (f_1, f_2) \in F(S)\}$$

For $\alpha \in [0, 1]$, let problem $LSPER(\alpha)$ be

$$maxf(w; \alpha) = \alpha f_2(w) + (1 - \alpha) f_1(w)$$

$$w \in S$$

and $S(\alpha, F) = \{(f_1, f_2) : \max_{f_1,f_2}\{\alpha f_2 + (1 - \alpha) f_1 : (f_1, f_2) \in F\}\}$. The elements of $S(\alpha, F)$ are called efficient solutions.

But, resolving the problem $LSPER(\alpha)$, only some efficient solutions are obtained. In order to obtain other solutions, we resolve the problem $LSPER^*$:

$$\min y$$

$$y \geq f_1 - g_1^*$$

$$y \geq g_2^* - f_2$$

$$x \in S$$

where $g_1^* = \min f_1$, $g_2^* = \max f_2$.

5.2 Example

Data are chosen so that as the machine gets older, the replacement and maintaining costs increase and the revenue decreases. Also, $n = 4$ $m = 2$, $J = 2$. The goal (g_1^*, g_2^*) is (28,33) and the efficient solutions obtained resolving the $LSPER(\alpha)$ are (28,33) and (32,33). Resolving the problem $LSPER^*$, another efficient solution is obtained (31,29).

5.3 Future work

In order to obtain every efficient solution another optimality criterion could be defined as the norm

$$\|\cdot\|_{(\alpha)} = \alpha\|\cdot\|_1 + (1-\alpha)\|\cdot\|_\infty$$

where $\|\cdot\|_1$ and $\|\cdot\|_\infty$ are Hölders l_p norms for $p = 1$ and $p = \infty$, respectively, and $\alpha \in [0,1]$ is a real number as above.

Also some improvements should be done for larger problems. The idea of constructing the valid inequalities for the problem $LSPER$ is under the consideration.

References

[1] G.H. Golub, C.F. Van Loan, 'Matrix Computations', 1989, John Hopkins Press

[2] . S. Hochbaum (edited by, 1997), 'Approximation Algorithms for NP-hard Problems', PWS Publishing Company, Boston

[3] A. H. G. Rinnooy Kan (1976), 'Machine Scheduling Problems', Classification, complexity and computations', Martinus Nijhoff, the Hague

[4] J. Shi and J. Malik, 'Normalized Cuts and Image Segmentation', Proc. *IEEE Conference on Computer Vision and Pattern Recognition*, 1997, pp. 731-737

[5] G. Sierksma (1996), 'Linear and Integer Programming', Marcel Dekker, Inc, New York, Basel

[6] K. Šorić, " The CLWS heuristics for Single Machine Scheduling Problem", *European Journal of Operational Research" ,* 1999, Vol. 120, No. 2, pp. 352-358

[7] K. Šorić, "A Cutting Plane Algorithm for Single Machine Scheduling Problem", *European Journal of Operational Research", ,* 2000, Vol. 127, No. 2, pp. 383-393

[8] K. Šorić, Z. Drmač, "A heuristics for One Machine Scheduling Problem", presentation on *IFORS 2002*, Edinburgh, July 8-12, 2002.

[9] K. Šorić, V. Vojvodić Rosenzweig, "Lot Sizing Problem with Equipment Replacement – Computational Results", presentation on 8th *International Conference on Operational Research,* Rovinj, Croatia, September 27 – 29, 2000

[10] Kristina Šorić, (1997), 'Exact Algorithms and Heuristics for Single Machine Scheduling Problem', PhD Thesis, Department of Pure and Applied Mathematics, University of Padova, Italy

[11] http://carbon.cudenver.edu/ hgreenbe/glossary/

[12] http://www.samizdat.mines.edu/ga_tutorial/

[13] http://www.mathematik.uni-osnabrueck.de/research/OR/class/

[14] http://www.nada.kth.se/ viggo/wwwcompendium/

[15] ttp://www.opsresearch.com/OR-Links/index.html

[16] ttp://www.personal.psu.edu/faculty/t/m/tmc7/tmclinks.html

[17] ttp://www.nada.kth.se/ viggo/wwwcompendium/node173.html