Local Search

Local search is an iterative algorithm that moves from one solution S to another S' according to some neighbourhood structure.

Local search procedure usually consists of the following steps.

1. Initialisation. Choose an initial schedule S to be the current solution and compute the value of the objective function $F(S)$.

2. Neighbour Generation. Select a neighbour S' of the current solution S and compute $F(S')$.

3. Acceptance Test. Test whether to accept the move from S to S'. If the move is accepted, then S' replaces S as the current solution; otherwise S is retained as the current solution.

4. Termination Test. Test whether the algorithm should terminate. If it terminates, output the best solution generated; otherwise, return to the neighbour generation step.

We consider four local search algorithms: Iterative Improvement, Threshold Accepting, Simulated Annealing, and Tabu Search. For all of them, steps 1, 2, and 4 are the same while step 3 is different.

We assume that a schedule is represented as a permutation of job numbers $(J_1, J_2, \dots , J_n)$. This can always be done for a single machine processing system or for permutation flow shop; for other models more complicate structures are used.

In Step 1, a starting solution can be obtained by one of the constructive heuristics described in the previous lectures or it can be specified by a random job permutation. If local search procedure is applied several times, then it is reasonable to use random initial schedules (*explain why*).

To generate a neighbour S' in Step 2, a neighbourhood structure should be specified beforehand. Often the following types of neighbourhoods are considered:

— *transpose neighbourhood* in which two jobs occupying adjacent positions in the sequence are interchanged: (1, **2**, **3**, 4, 5, 6, 7) → (1, **3**, **2**, 4, 5, 6, 7);

— *swap neighbourhood* in which two arbitrary jobs are interchanged:
(1, **2**, 3, 4, 5, **6**, 7) → (1, **6**, 3, 4, 5, **2**, 7);

— *insert neighbourhood* in which one job is removed from its current position and inserted elsewhere: (1, **2**, 3, 4, 5, 6, 7) → (1, 3, 4, 5, 6, **2**, 7).

Neighbours can be generated randomly, systematically, or by some combination of the two approaches.

In Step 3, the acceptance rule is usually based on values $F(S)$ and $F(S')$ of the objective function for schedules S and S'. In some algorithms only moves to 'better' schedules are accepted (schedule S' is better than S if $F(S')<F(S)$); in others it may be allowed to move to 'worse' schedules. Sometimes "wait and see" approach is adopted.

The algorithm terminates in Step 4 if the computation time exceeds the prespecified limit or after completing the prespecified number of iterations.

In what follows we specify Step 3 "Acceptance Test" for each type of the local search algorithm.

1 Iterative Improvement

Iterative Improvement allows only strict improvement in the objective function value.

— It accepts a new schedule *S'* only if *F*(*S'*)<*F*(*S*), where *S* is the current schedule.

Often instead of accepting the first neighbour with the value of the objective function smaller than *F*(*S*) for the current schedule *S*, the algorithm constructs all neighbours (or a given number of neighbours) and selects the best one.

The algorithm stops when for all neighbours *S'* of schedule *S*, *F*(*S'*)≥*F*(*S*), i.e., when a local optimum is obtained. A better schedule may be found if the algorithm is applied repeatedly starting with different randomly generated initial solutions.

<u>*In-class exercise 1*</u> (from "Scheduling: Theory, Algorithms and Systems" by M. Pinedo)

Consider the following instance of problem $1||\Sigma w_j T_j$ (single machine scheduling problem of minimising total weighted tardiness).

Jobs	p_j	d_j	w_j
1	10	4	14
2	10	2	12
3	13	1	1
4	4	12	12

Apply Iterative Improvement algorithm starting with initial schedule S_1=(4,3,2,1). Define the neighbourhood as all schedules that can be obtained from a current schedule through ***adjacent pairwise interchanges***. In each iteration consider ***all*** neighbours of the current schedule.

Current schedule, $\Sigma w_j T_j$	Neighbour, $\Sigma w_j T_j$	Accepted?

2 Threshold Accepting

Threshold Accepting allows to continue the local search even if a local optimum has been obtained.

— It accepts a new schedule *S'* if $F(S') < F(S) + \alpha$, where $\alpha > 0$ is a threshold value.

Usually α is relatively large at the beginning and it becomes smaller later on.

3 Simulated Annealing

Simulated Annealing also allows to continue the local search even if a local optimum has been obtained. It uses the ***Probabilistic Acceptance Test***, which can be described as follows.

Determine $\Delta = F(S') - F(S)$.

— If $\Delta \leq 0$, then a move to schedule *S'* is always accepted.

— If $\Delta > 0$, then a move to schedule *S'* is accepted with probability $e^{-\Delta/T}$, where *T* is a parameter called the *temperature*, which changes during the course of the algorithm.

Usually *T* is large in the beginning and then it decreases until it is close to 0 at the final stages. Different "*cooling schemes*" can be applied. Often $T_k = a^k$, where T_k is the "temperature" at iteration *k* and $0 < a < 1$.

Both algorithms, Threshold Accepting and Simulated Annealing, can get back to the solutions already visited and this is their main disadvantage. A simple way to avoid cycling is to store visited solutions in a list called a tabu list. A new solution can be accepted if it is not contained in the list.

4 Tabu Search

Tabu Search algorithm allows accepting a "worse" schedule *S'* (as Threshold Accepting and Simulated Annealing). Its acceptance test is based on a *tabu list*. Tabu list stores attributes of the previous few moves. It has a fixed number of entries (usually between 5 and 9) and it is updated each time a new schedule *S'* is accepted:

— the reverse transformation is entered at the top of the tabu list to avoid returning to the same solution (to avoid returning to a local optimum);

— all other entries are pushed down one position;

— the bottom entry is deleted.

The following ***Deterministic Acceptance Test*** is usually implemented.

Determine $\Delta = F(S') - F(S)$.

— If $\Delta < 0$ and *S'* is "non-tabu", then a move to *S'* is always accepted.

— If $\Delta < 0$ and *S'* is "tabu", then a move to *S'* may be accepted for a "promising" schedule *S'* (if $F(S')$ is less than the objective function value for any other solution obtained before).

— If $\Delta \geq 0$ and *S'* is "tabu", then a move to *S'* is always rejected.

— If $\Delta \geq 0$ and *S'* is "non-tabu", then a "wait and see" approach is adopted: *S'* remains as a candidate while the search continues for a neighbour which can be accepted immediately. If no such neighbour is found, a move to the best candidate *S'* is made.

In-class exercise 2 (from “Scheduling: Theory, Algorithms and Systems” by M. Pinedo)
Consider the same instance of problem $1||\Sigma w_j T_j$. Apply Tabu Search starting with initial schedule S_1=(4,2,3,1). Define the neighbourhood as the schedules that can be obtained from a current schedule through ***adjacent pairwise interchanges***. Accept the first non-tabu neighbour with $\Delta<0$, if one exists; otherwise consider all neighbours of the current schedule. Assume that tabu-list is a list of pairs of jobs (*j, k*) that were swapped within the last two moves and cannot be swapped again.

Tabu List	Current schedule, $\Sigma w_j T_j$	Neighbour, $\Sigma w_j T_j$	Accepted?

Conclusions

1. Local search algorithms are very generic.
2. They have been applied successfully to many industrial problems.
3. Performance of local search algorithms depends on construction of neighbourhood.
4. A method that exploits the special structure of a particular problem is usually faster (if one exists).

Log Book - List of Schedules.seq

Schedule	Time	C_{max}	T_{max}	$\sum U_j$	$\sum C_j$	$\sum T_j$	$\sum w_j C_j$	$\sum w_j T_j$
Sch 1234	1	37	32	4	100	81	857	632
Sch 1243	1	37	36	4	91	72	705	480
Sch 1324	1	37	31	4	103	84	1003	778
Sch 1342	1	37	35	4	97	78	931	706
Sch 1423	1	37	36	4	85	66	633	408
Sch 1432	1	37	35	4	88	69	779	554
Sch 2134	1	37	32	4	100	81	877	652
Sch 2143	1	37	36	4	91	72	725	500
Sch 2314	1	37	29	4	103	84	1049	824
Sch 2341	1	37	33	4	97	78	985	760
Sch 2413	1	37	36	4	85	66	661	436
Sch 2431	1	37	33	4	88	69	833	608
Sch 3124	1	37	31	4	106	87	1175	950
Sch 3142	1	37	35	4	100	81	1103	878
Sch 3214	1	37	29	4	106	87	1195	970
Sch 3241	1	37	33	4	100	81	1131	906
Sch 3412	1	37	35	4	94	75	1039	814
Sch 3421	1	37	33	4	94	75	1059	834
Sch 4123	1	37	36	3	79	68	569	440
Sch 4132	1	37	35	3	82	71	715	586
Sch 4213	1	37	36	3	79	68	589	460
Sch 4231	1	37	33	3	82	71	761	632
Sch 4312	1	37	35	3	85	74	887	758
Sch 4321	1	37	33	3	85	74	907	778