DM87
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 6

Local Search Heuristics, Exercises

Marco Chiarandini

Outline

2.1. Consider the instance of $1 \mid\mid \sum w_j C_j$ with the following processing times and weights.

jobs	1	2	3	4
w_j	6	11	9	5
p_j	3	5	7	4

(a) Find the optimal sequence and compute the value of the objective.
(b) Give an argument for positioning jobs with larger weight more toward the beginning of the sequence and jobs with smaller weight more toward the end of the sequence.
(c) Give an argument for positioning jobs with smaller processing time more toward the beginning of the sequence and jobs with larger processing time more toward the end of the sequence.
(d) Determine which one of the following two generic rules is the most suitable for the problem:
i. sequence the jobs in decreasing order of $w_j - p_j$;
ii. sequence the jobs in decreasing order of w_j/p_j.

2.2. Consider the instance of $1 \mid\mid L_{\max}$ with the following processing times and due dates.

jobs	1	2	3	4
p_j	5	4	3	6
d_j	3	5	11	12

(a) Find the optimal sequence and compute the value of the objective.
(b) Give an argument for positioning jobs with earlier due dates more toward the beginning of the sequence and jobs with later due dates more toward the end of the sequence.
(c) Give an argument for positioning jobs with smaller processing time more toward the beginning of the sequence and jobs with larger processing time more toward the end of the sequence.
(d) Determine which one of the following four rules is the most suitable generic rule for the problem:
i. sequence the jobs in increasing order of $d_j + p_j$;
ii. sequence the jobs in increasing order of $d_j p_j$;
iii. sequence the jobs in increasing order of d_j;
iv. sequence the jobs in increasing order of p_j.

Outline

Software Tools

- Modeling languages
  interpreted languages with a precise syntax and semantics
- Software libraries
  collections of subprograms used to develop software
- Software frameworks
  set of abstract classes and their interactions
  - *frozen spots* (remain unchanged in any instantiation of the framework)
  - *hot spots* (parts where programmers add their own code)

No well established software tool for Local Search:

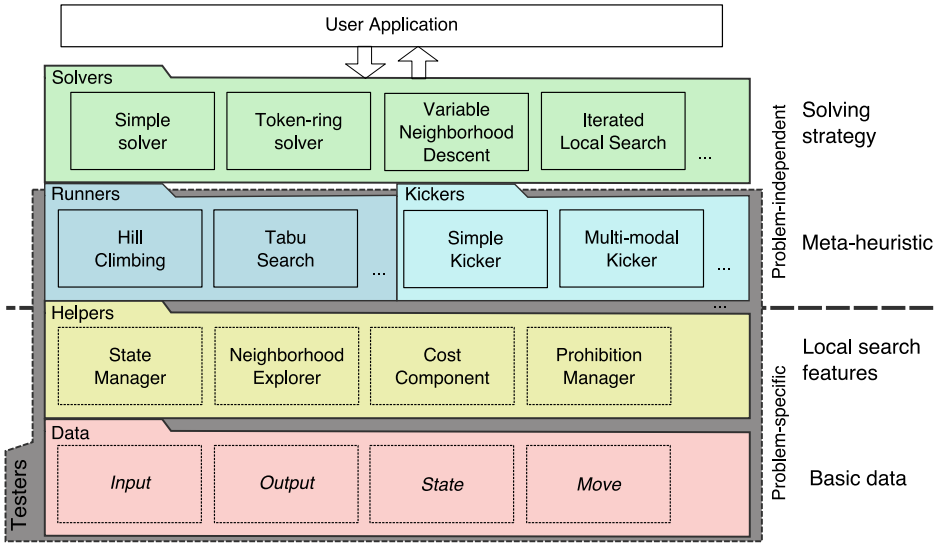
- the apparent simplicity of Local Search induces to build applications from scratch.
- crucial roles played by delta/incremental updates which is problem dependent
- the development of Local Search is in part a craft, beside engineering and science.
- lack of a unified view of Local Search.

Software tools for Local Search and Metaheuristics

Tool	Reference	Language	Type
ILOG	[Shaw et al., 2002]	C++, Java, .NET	LS
GAlib	[Wall, 1996]	C++	GA
GAUL	[Adcock, 2005]	C	GA
Localizer++	[Michel and Van Hentenryck, 2000]	C++	Modeling
HotFrame	[Fink and Voß, 2002]	C++	LS
EasyLocal++	[Di Gaspero and Schaerf, 2003]	C++, Java	LS
HSF	[Dorne and Voudouris, 2004]	Java	LS, GA
ParadisEO	[Cahon et al., 2004]	C++	EA, LS
OpenTS	[Harder et al., 2004]	Java	TS
MDF	[Lau et al., 2007]	C++	LS
TMF	[Watson, 2007]	C++	LS
SALSA	[Laburthe and Caseau, 2002]	—	Language
Comet	[Van Hentenryck and Michel, 2005]	—	Language

table prepared by L. Di Gaspero

Separation of Concepts in Local Search Algorithms



implemented in EasyLocal++

Outline

Input (util.h, util.c)

```
typedef struct {
  long int number_jobs; /* number of jobs in instance */
  long int release_date[MAX_JOBS]; /*there is no release date for these instances*/
  long int proc_time[MAX_JOBS];
  long int weight[MAX_JOBS];
  long int due_date[MAX_JOBS];
} instance_type;

instance_type instance;

void read_problem_size (char name[100])
void read_instances (char input_file_name[100])
```


State/Solution (util.h)

```
typedef struct {
  long int job_at_pos[MAX_JOBS]; /* Gives the job at a certain pos */
  long int pos_of_job[MAX_JOBS]; /* Gives the position of a specific job */
  long int completion_time_job[MAX_JOBS]; /* Gives C_j of job j */
  long int start_time_job[MAX_JOBS]; /* Gives start time of job j */
  long int tardiness_job[MAX_JOBS]; /* Gives T_j of job j */
  long int value; /* Objective function value */
} sol_representation;

sol_representation sequence;
```

Output (util.c)

```
void print_sequence (long int k)
void print_completion_times ()
```

State Manager (util.c)

```
void construct_sequence_random ()
void construct_sequence_canonical ()
long int evaluate ()
```

Random Generator (random.h, random.c)

```
void set_seed (double arg)
double MRG32k3a (void)
double ranU01 (void)
int ranUint (int i, int j)
void shuffle (int *X, int size)
```

Timer (timer.c)

```
double getCurrentTime ()
```


Outline

Your Task on $1||\sum_j w_j T_j$

1. Implement two basic local search procedures that return a local optimum:

```
void ls_swap_first( ) {};
void ls_interchange_first( ) {};
```

2. Implement the other neighborhood for permutation representation mentioned at the lecture from one of the two previous neighborhoods.
3. Provide computational analysis of the LS implemented. Consider:
   - size of the neighborhood
   - diameter of neighborhood
   - complete neighborhood examination
   - local optima attainment
4. Devise speed ups to reduce the computational complexity of the LS implemented
5. Improve your heuristic in order to find solutions of better quality. (Hint: use a construction heuristic and/or a metaheuristic)

Adcock, S. (2005).
Genetic algorithms utitlity library.
Web Page.

Cahon, S., Melab, N., and Talbi, E. G. (2004).
ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics.
Journal of Heuristics, 10(3):357–380.

Di Gaspero, L. and Schaerf, A. (2003).
EasyLocal++: An object-oriented framework for flexible design of local search algorithms.
Software — Practice & Experience, 33(8):733–765.

Dorne, R. and Voudouris, C. (2004).
Hsf: the iopt's framework to easily design metaheuristic methods.
pages 237–256.

Fink, A. and Voß, S. (2002).
HotFrame: A heuristic optimization framework.
In *Optimization Software Class Libraries*, pages 81–154. Kluwer Academic Publishers.


Harder, R., Hill, R., and Moore, J. (2004).
A java universal vehicle router for routing unmanned vehicles.
International Transactions in Operations Research, 11:259–275.


Laburthe, F. and Caseau, Y. (2002).
Salsa: A language for search algorithms.
Constraints, 7(3-4):255–288.


Lau, H. C., Wan, W. C., Halim, S., and Toh, K. Y. (2007).
A software framework for rapid hybridization of meta-heuristics.
International Transactions in Operations Research, 14(2).


Michel, L. and Van Hentenryck, P. (2000).
Localizer.
Constraints, 5(1–2):43–84.


Shaw, P., De Backer, B., and Furnon, V. (2002).
Improved local search for cp toolkits.
Annals of Operations Research, 20(1–4):31–50.


Van Hentenryck, P. and Michel, L. (2005).
Constraint-based Local Search.
MIT Press.


Wall, M. B. (1996).
A Genetic Algorithm for Resource-Constrained Scheduling.
PhD thesis, MIT Mechanical Engineering Department.


Watson, J.-P. (2007).
The templatized metaheuristic framework.
In *Proceedings of the 7th Metaheuristics International Conference (MIC 2007)*.

DM87 – Scheduling, Timetabling and Routing 15