Local Search Heuristics, Exercises

Marco Chiarandini

Outline

1. An Overview of Software for LS Methods
2. The Code Delivered
3. Practical Exercise

Software Tools

- 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)
- Modeling languages
  interpreted languages with a precise syntax and semantics
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.

<table>
<thead>
<tr>
<th>Software tools for Local Search and Metaheuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tool</strong></td>
</tr>
<tr>
<td>ILOG</td>
</tr>
<tr>
<td>GAlib</td>
</tr>
<tr>
<td>GAUL</td>
</tr>
<tr>
<td>Localizer++</td>
</tr>
<tr>
<td>HOTFRAME</td>
</tr>
<tr>
<td>EASYLOCAL++</td>
</tr>
<tr>
<td>HSF</td>
</tr>
<tr>
<td>ParadisEO</td>
</tr>
<tr>
<td>OpenTS</td>
</tr>
<tr>
<td>MDF</td>
</tr>
<tr>
<td>TMF</td>
</tr>
<tr>
<td>SALSA</td>
</tr>
<tr>
<td>Comet</td>
</tr>
</tbody>
</table>

*Table prepared by L. Di Gaspero*

Outline

1. An Overview of Software for LS Methods
2. The Code Delivered
3. Practical Exercise

Separation of Concepts in Local Search Algorithms

implemented in EASYLOCAL++
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

1. An Overview of Software for LS Methods
2. The Code Delivered
3. Practical Exercise
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 search 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)

References: