Metaheuristics: a quick overview

Marc Sevaux
University of Valenciennes
CNRS, UMR 8530, LAMIH / Production systems
Marc.Sevaux@univ-valenciennes.fr
Outline

- Neighborhood metaheuristics
- Application on a parallel machine scheduling problem
  \[ Pm | r_j | \sum w_j U_j \]
- Conclusion on neighborhood search methods
- Population metaheuristics
- Application on a single machine scheduling problem
  \[ 1 | r_j | \sum w_j T_j \]
- Conclusion on population metaheuristics
Motivations for neighborhood metaheuristics

- \(NP\)-hard problems
- Obtain rapidly solutions
- Short time of implementation
- No/few knowledge of the problem
- ... it’s easier...

**Notation**

- a set of solutions, \(S\)
- a function to minimize, \(f(s) : s \in S \rightarrow IR\)
- an optimal solution, \(s^* \in S / f(s^*) \leq f(s) \ \forall s \in S\)
Neighborhood and initial solution

First, an initial solution is needed. Then, to explore a solution space, we need to know how to move from one solution to another. It’s the neighborhood.

- Find an initial solution
- Define a neighborhood
- Explore [exhaustively] the neighborhood
- Evaluate a neighbor.

Notation

- a neighborhood of $s$, $N(s)$
Different approaches

Descent methods

- Simple descent, deepest descent
- Multi-start

Metaheuristic methods

- TS : Tabu search [Glover, 89 et 90]
- SA : Simulated annealing [Kirckpatrick, 83]
- TA : Threshold accepting [Deuck, Scheuer, 90]
- VNS : Variable neighborhood [Hansen, Mladenović, 98]
- ILS : Iterated local search [Lorenço et al, 2000]

Practical use in scheduling

- Deepest descent, Multi-start
- Simulated annealing, Tabu search
Simple descent

procedure Simple_Descent(initial solution $s$)
    repeat
        Choose $s' \in N(s)$
        if $f(s') < f(s)$ then
            $s \leftarrow s'$
        end if
    until $f(s') \geq f(s)$, $\forall s' \in N(s)$
end
Simple descent

![Graph showing simple descent with initial solution and final solution in the solution space.](image)
Deepest descent

procedure Deepest_Descent(initial solution \( s \))
   repeat
      Choose \( s' \in N(s) / f(s') \leq f(s'') \forall s'' \in N(s) \)
      if \( f(s') < f(s) \) then
         \( s \leftarrow s' \)
      end if
   until \( f(s') \geq f(s), \forall s' \in N(s) \)
end
Deepest descent

Initial solution

Final solution

Objective value

Solution space
Multi-start descent and deepest descent

procedure Multistart_Descent

iter = 1
f(Best) = +∞
repeat
  Choose a starting solution $s_0$ at random
  $s \leftarrow$ Simple_Descent($s_0$) (ou Deepest_Descent($s_0$))
  if $f(s) < f(Best)$ then
    Best $\leftarrow$ s
  end if
  iter $\leftarrow$ iter + 1
until iter = IterMax

end
Multi-start descent and deepest descent

Initial solutions

Best solution

Final solutions

Solution space

Objective value
Tabu search

**procedure** Tabu-Search(*initial solution* s)

\[ Best \leftarrow s \]

**repeat**

Choose \( s' \in N(s) / f(s') \leq f(s'') \ \forall s'' \in N(s) \)

**if** \( s' \) is not tabu \textbf{or} \( f(s') < f(Best) \) **then**

\[ s \leftarrow s' \]

Update tabu list, update *Best*

**end if**

**until** stopping conditions are met

**end**
Tabu search

Initial solution

Final solution

Best solution

Objective value

Solution space
Simulated annealing

procedure Simulated_Annealing(initial solution $s$)

$Best \leftarrow s$

$T \leftarrow T_{init},$ choose $\alpha \in ]0, 1[$, choose $IterStage$

repeat

for $i = 1$ to $IterStage$

Choose $s' \in N(s)$

$\Delta \leftarrow f(s') - f(s)$

$s \leftarrow \begin{cases} 
  s' & \text{if } \Delta < 0 \text{ or } \\
  \text{with the probability } \exp \frac{-\Delta}{T} & \text{ Update Best}
\end{cases}$

end for

$T \leftarrow \alpha \times T$

until stopping conditions are met

end
Simulated annealing

![Diagram of Simulated Annealing]

- **Solution space**
- **Objective value**
- **Initial solution**
- **Best solution**
- **Final solution**
Threshold accepting

procedure Threshold(initial solution $s$)

$T \leftarrow T_{\text{init}}$, choose $\alpha \in ]0, 1[$, choose IterStage

$Best \leftarrow s$, moved $\leftarrow$ true

while moved do

for $i = 1$ to IterStage

Choose $s' \in N(s)$, $\Delta \leftarrow f(s') - f(s)$

moved $\leftarrow$ true

if $\Delta < 0$ or $\Delta < T$ then $s \leftarrow s'$

else moved $\leftarrow$ false, end if

Update Best

end for

$T \leftarrow \alpha \times T$

end while

end
Threshold accepting

- Solution space
- Objective value
- Initial solution
- Best solution
- Final solution
Variable neighborhood search

procedure VNS(initial solution s)
    Best ← s
    Choose a set of neighborhood structures \( N_k \) \( k = 1 \ldots k_{max} \)
    repeat
        \( k ← 1 \)
        repeat
            Shaking: choose \( s' \in N_k(s) \)
            \( s'' ← Localsearch(s') \)
            if \( f(s'') < f(s) \) then \( s ← s'' \)
            else \( k ← k + 1 \) end if
            Update Best
        until \( k = k_{max} \)
    until stopping conditions are met
end
Variable neighborhood search

Initial solution (s)

Objective value

Solution space

Marc Sevaux

TEW – Antwerp 2003
Iterated local search

procedure $ILS$(*initial solution* $s_0$)

$Best \leftarrow s_0$
$s \leftarrow Localsearch(s_0)$

repeat

$s' \leftarrow Perturbation(s, history)$
$s'' \leftarrow Localsearch(s')$
$s \leftarrow AcceptanceCriterion(s, s'', history)$

Update $Best$

until stopping conditions are met
end
Iterated local search

Initial solution $s_0$

Objective value

Solution space

Iterated local search

Perturbation
References


[Deuck et Scheuer 90 ] Threshold accepting: a general purpose optimization algorithm... Journal of Computational Physics 90, pp161-175.


Conclusion on neighborhood search methods

These metatheuristics

- are easy to understand and control
- give solutions rapidly
- need a short time for implementation
- and few knowledge of the problem

But to be really efficient

- tricks have to be added
- we need to know the problem very well
- try many of them
- and why not combine them
Population metaheuristics

- Based on a population of solutions
- How to combine different solutions?
- How to manage the population?
- How to evaluate a solution?
Different approaches

• Genetic Algorithm, Holland 1975 – Goldberg 1989
• Memetic Algorithm, Moscatto 1989
• Hybrid Genetic Algorithm
• Ant Colony Optimisation, Dorigo 1991
• Scatter search, Laguna, Glover, Martì 2000
• Application to scheduling
Genetic algorithm

Generate an initial population

While stopping conditions are not met
  Select two individuals
  Apply crossover operator
  Mutate offspring under probability
  Insert offspring under conditions
  Remove an individual under conditions

End While
Memetic algorithm

Generate an initial population
While stopping conditions are not met
    Select two individuals
    Apply crossover operator
    Improve offspring under probability
    Insert offspring under conditions
    Remove an individual under conditions
End While
Hybrid genetic algorithm

Generate an initial population

While stopping conditions are not met

Select two individuals

Apply crossover operator

Improve offspring under probability

Mutate offspring under probability

Insert offspring under conditions

Remove an individual under conditions

End While
Description of the algorithms

Ant Colony Optimisation

Nest -> Food

Obstacle

Nest -> Food

Obstacle

Nest -> Food

Obstacle

Nest -> Food

Obstacle

Marc Sevaux

TEW – Antwerp 2003
Ant Colony Optimisation

Generate an initial colony
While stopping conditions are not met
    For each ant of the colony
        initialize ant memory
        While Current State ≠ Target state
            Apply ant decision under pheromone information
            Move to next state
            Deposit pheromone information on the transition
        End While
    End For
End While
Evaporate pheromone information
End While
Scatter search

Generate an initial improved population
Select a diverse subset ($RefSet$, size $b$)

While stopping conditions are not met

\[ A \leftarrow RefSet \]

While $A \neq \emptyset$

\[ Combine \ solutions \ (B \leftarrow RefSet \times A) \]
\[ Improve \ solutions \]
\[ Update \ RefSet \ (keep \ b \ best \ from \ RefSet \cup B) \]
\[ A \leftarrow B - RefSet \]

End While

Remove the worst $b/2$ solutions from $RefSet$
Add $b/2$ diverse solutions to $RefSet$

End While
## Intensification vs diversification

<table>
<thead>
<tr>
<th></th>
<th>Intensification</th>
<th>Diversification</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>selection, crossover, replacement</td>
<td>mutation</td>
</tr>
<tr>
<td>MA</td>
<td>selection, crossover, local search, replacement</td>
<td></td>
</tr>
<tr>
<td>HGA</td>
<td>selection, crossover, local search, replacement</td>
<td>mutation</td>
</tr>
<tr>
<td>ACO</td>
<td>pheromone information</td>
<td>colony, ant memory re-initialization</td>
</tr>
<tr>
<td>SS</td>
<td>inner loop: crossover, local search</td>
<td>diverse replacement</td>
</tr>
</tbody>
</table>
References (web sites)

http://www-illigal.ge.uiuc.edu/index.php3

Ant Colony Optimisation  Dorigo 1991
http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html

Memetic Algorithms  Moscatto 1989
http://www.densis.fee.unicamp.br/
   ~moscato/memetic_home.html

Scatter search  Laguna, Glover, Martì 2000
http://www-bus.colorado.edu/faculty/
   laguna/scattersearch.htm
References

Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, 1999

Ant Colony Optimisation  Dorigo 1991
References

Memetic Algorithms Moscatto 1989

Scatter search Laguna, Glover, Martí 2000
Application to a scheduling problem

Scheduling Application
Minimizing the total weighted tardiness on a single machine in the general case, \( 1|r_j|\sum w_jT_j \)

Aim of the study
Point out the differences between an Hybrid Genetic Algorithm and a Scatter Search

Implementation
Both methods are compared on a same basis (common components, identical stopping conditions, equivalent evaluations).
Previous work

**Complexity status**

\[ 1\mid r_j \mid \sum w_j T_j \] is \( \mathcal{NP} \)-Hard in a strong sense

\[ 1\mid r_j \mid \sum T_j \] \( \mathcal{NP} \)-Hard in a strong sense [Garey, Johnson 77]

\[ 1\mid \sum w_j T_j \] \( \mathcal{NP} \)-Hard in a strong sense [Lawler 77]

\[ 1\mid \sum T_j \] \( \mathcal{NP} \)-Hard in an ordinary sense [Du, Leung 90]

**Recent approaches** on \( 1\mid \sum w_j T_j \) (OR-Library)

[Crauwels, Potts, van Wassenhove 98] SA, TS, GA

[Congram, Potts, van de Velde 02] Iterated dynasearch algorithm

**Other approaches** on \( 1\mid r_j \mid \sum w_j T_j \)

[Akturk, Ozdemir 00] Exact method (up to 20 jobs)

[Jouglet 02] Constraint-based method (up to 40 jobs)
Common components

Coding
Permutation = natural representation for single machine scheduling problems, no clones allowed

Fitness
Build semi-active schedules according to the order of the permutation

Crossover
Standard LOX crossover

Mutation
GPI: General Pairwise Interchange

Local search
Based on the GPI, a best improvement method is applied
Parameters (choose or tune)

Chosen according to the model

- crossover operator
- mutation operator
- local search method
- distance (for diverse replacement method)

Tuned according to the problem

- mutation rate
- local search rate
- population/RefSet size
- stopping conditions
What could happen

• Easy problems (SS)
  cannot generate a diverse population

• Size of the population (GA/SS)
  SS: Too large → too slow
  GA: Too small → not enough diversification

• Mutation rate (GA)
  Too low → cannot escape from a local optima
  Too high → cannot converge

• Local search rate (GA)
  Too low → converges slowly
  Too high → converges too quickly to a local optima
Influence of mutation rate
What could happen

Influence of local search rate

Marc Sevaux
TEW – Antwerp 2003
What could happen

Extreme rate values
Numerical results

Several sets of instances are used for numerical evaluation

**OR-LIB** instances for $1|\sum w_j T_j$

$n \in \{40, 50, 100\}$, 125 instances each

Comparison to optima (when known) or best solutions

**ODD** instances for $1|r_j|\sum w_j T_j$ (initially for $1|r_j|\sum w_j U_j$)

$n \in \{20, 40, 60, 80, 100\}$, 20 instances each

Comparison between GA and SS approaches
Generic parameters

Parameters will be arbitrarily fixed for the study, better solution can be found by tuning the parameters correctly.

Population size (SS) NPop = 10

RefSet size $b = 5$

Population size (GA) NPop = $n/2$

Mutation rate $mr = 0.1$

Local search rate $lsr = 0.1$

Stopping conditions 1200 sec. or 10000 it. without improvement
(for ORLIB_100 jobs: 120 sec.)
## OR LIB results

Parameters need to be tuned more precisely

<table>
<thead>
<tr>
<th>Instances</th>
<th>CPU time</th>
<th>GA</th>
<th>it. SS</th>
<th>GA = SS</th>
<th>GA &gt; SS</th>
<th>Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORLIB_40</td>
<td>3.89s</td>
<td></td>
<td>4836</td>
<td>119</td>
<td>6</td>
<td>125</td>
</tr>
<tr>
<td>ORLIB_50</td>
<td>8.16s</td>
<td></td>
<td>2349</td>
<td>0</td>
<td>125</td>
<td>124</td>
</tr>
<tr>
<td>ORLIB_100*</td>
<td>58.36s</td>
<td></td>
<td>2</td>
<td>42</td>
<td>62</td>
<td>54</td>
</tr>
</tbody>
</table>

*CPU time limit is fixed to 120s

GA is always limited by the limit of 10,000 it w/o improvement
SS is always limited by the CPU time limit
Another set of instances ODD

A problem instance generator is used, it is based on a single day of 80 periods. Twenty instances are generated for each value of $n$ ($n \in \{20, 40, 60, 80, 100\}$).

Orders have a high probability of arriving in the morning. Due dates are in the afternoon with high probability.

Release dates distributed according to a Gamma law $\Gamma(0.2, 4)$
(Mean $= 20$, St. dev. $= 10$)

Due dates distributed according to the same Gamma law (Mean $= 20$, St. dev. $= 10$) but starting at the end of the horizon

Processing times uniformly distributed in $[1, d_i - r_i]$

Weights uniformly distributed in $[1, 10]$
Release date and due date repartition

![Graph showing release dates and due dates distribution over time periods]

Numerical results
## ODD results

Stopping conditions:
- CPU time limit = 1200s and 10,000 it. w/o improvement

<table>
<thead>
<tr>
<th>Inst.</th>
<th>CPU GA</th>
<th>it. SS</th>
<th>GA best</th>
<th>SS best</th>
<th>GA = SS</th>
<th>GA &gt; SS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODD.20*</td>
<td>13.34s</td>
<td>8610</td>
<td>0.4s</td>
<td>0.0s</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>ODD.40</td>
<td>5.60s</td>
<td>1253</td>
<td>0.6s</td>
<td>3.1s</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>ODD.60</td>
<td>21.21s</td>
<td>291</td>
<td>4.6s</td>
<td>19.1s</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>ODD.80</td>
<td>62.68s</td>
<td>99</td>
<td>9.2s</td>
<td>248.9s</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>ODD.100</td>
<td>111.16s</td>
<td>43</td>
<td>33.6s</td>
<td>289.0s</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

*3 instances cannot be solved by SS, cannot generate 10 diverse solutions

GA is always limited by the limit of 10,000 it w/o improvement
SS is always limited by the CPU time limit
Conclusion

GA and SS are both very powerful with similar kind of efforts

Hybrid Genetic Algorithm

- “easy” to implement
- lot of parameters to be tuned
- different class of instances $\rightarrow$ different parameters
- need automatic parameter configuration procedures

Scatter Search Algorithm

- “somewhat harder” to implement
- few parameters to be tuned
- need to improve the local search procedure