An Effective Evolutionary Hybrid for Solving the Permutation Flowshop Scheduling Problem

Amirghasemi & Zamani (2015)

Finn van der Heide, Job Hartjes, Mick van het Nederend
Content

● The problem
● The algorithm
● Results
● Conclusion
● Discussion
Introduction

- Local search: *local optima*
- Genetic algorithms: *hitchhiking*
- Combination (memetic algorithm)
Problem description 1

The permutation flowshop scheduling problem

- Control flow
- Machines $p_1, p_2, ..., p_n$
- Jobs $J_1, J_2, ..., J_m$
- Each job has $n$ operations ($i$'th operation on $i$'th machine)

Goal: Minimize idle time
Problem description 1

The permutation flowshop scheduling problem

- Control flow
- Machines $p_1, p_2, \ldots, p_n$
- Jobs $J_1, J_2, \ldots, J_m$
- Each job has $n$ operations ($i$'th operation on $i$'th machine)

Goal: Minimize idle time
Problem description 2

$C_{\text{max}} = 34$

Problem description 3 - critical path

\[ \begin{array}{cccc}
M_1 & M_1 & M_1 & M_1 \\
J_1 & 5 & 4 & 4 & 3 \\
J_2 & 5 & 4 & 4 & 6 \\
J_3 & 5 & 5 & 5 & 3 \\
J_4 & 5 & 5 & 5 & 2 \\
J_5 & 5 & 5 & 5 & 5 \\
\end{array} \]
Problem description 3 - critical path
Problem description 3 - critical path
Problem description 3 - critical path
Problem description 3 - critical path

\[ C_{\text{max}} = 34 \]

Problem description 4

Goal: Minimize idle time

Rearrange order of jobs in such a way that the cumulative time of the critical path is minimized.

<table>
<thead>
<tr>
<th>Job</th>
<th>$P_1, J_1$</th>
<th>$P_2, J_1$</th>
<th>$P_3, J_1$</th>
<th>$P_4, J_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_2$</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$J_3$</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>$J_3$</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$J_4$</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$J_5$</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
Reblocking Mechanism 1

- Most likely blocks to be used for intermediates for connecting the beginning to the end.
- Calculate the total time of starting, middle and ending operations of each job.
- Sort the jobs in ascending order of total times of each block.

```
<table>
<thead>
<tr>
<th>Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
</tr>
<tr>
<td>values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Jobs in a given order (solution)</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
</tr>
<tr>
<td>values</td>
</tr>
</tbody>
</table>

```

```

<table>
<thead>
<tr>
<th>Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

```

```

<table>
<thead>
<tr>
<th>Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

```

```
Reblocking Mechanism 2

- Northeast and Southwest can’t be connected.
- Calculate the total time of starting, middle and ending operations of each job.
- Sort the jobs in descending order of total times for the northeast block.
- Sort the jobs in ascending order of total times for the southwest block.
Reblocking Construction

- Reblocking Mechanism 2 was discarded since the first one was 2.24% better.

To create an initial solution

- From the $t$ best possibilities for the current block a random one is chosen.
- This is done till the permutation is filled.
Generation

- Randomly take two members of the population.
- Create offspring (kX)
- Do variable neighborhood search on the offspring
  - Swap local search
  - Insertion local search
- Use truncation selection (top 50%) on $\mu + \lambda$

Parent selection highly favors diversification.

Survival selection highly favors intensification.
Modified K-point Crossover

- Create k+1 sections.
- Odd and even sections are identified.
- First offspring takes the odd sections of the first parent.
- Rest is filled in the order of the second parent.

<table>
<thead>
<tr>
<th>Parent 1</th>
<th>4</th>
<th>1</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>6</th>
<th>9</th>
<th>8</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent 2</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>Offspring 1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Offspring 2</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>
Swapping

- Sort on total duration on the critical path.
- Try all swaps until one that is beneficial is found.

Not all swaps need exact evaluation

- Locations on the original critical path are stored.
- Critical path on the new arrangement is not changed.
- Result is evaluated and rejected if not better.
Local Search

- Both local searches are done in succession.
- Until no improvement is found.

Optimisation

- When the jobs have their critical operations on the same machine, then the swap/insertion is ignored.
RAMP performance comparison

- $\text{NEGA}_{\text{VNS}}^1$, a genetic algorithm that uses the NEH$^2$ for initial solutions
- $\text{PSO}_{\text{VNS}}^3$, a particle swarm optimization algorithm
- $\text{SAOP}^4$, a simulated annealing algorithm
- $\text{PACO}^5$ and $\text{M-MMAS}^5$, two ant colony optimization algorithms
- $\text{ILS}^6$, iterated local search
- $\text{HGA\_RMA}^7$, a hybrid genetic algorithm

1 Zobolas et al., 2009
2 Nawaz, Enscore, and Ham, 1983
3 Tasgetiren et al., 2007
4 Osman and Potts, 1989
5 Rajendran and Ziegler, 2004
6 Stützle, 1998
7 Ruiz et al., 2006
RAMP performance comparison

Average percent deviation from the best-known solutions

Graph showing the comparison of RAMP and other algorithms with the best-known solutions.
RAMP performance comparison

Average time needed to reach the best solution

PSOVNS
NEGAVNS
RAMP
RAMP vs NEGA\textsubscript{VNS}

- The results indicate that the RAMP performed comparatively and that for some initial random seeds, it was able to produce high-quality solutions in a very short amount of time.

- Out of 120 instances:
  - In 31 instances the RAMP obtained better results
  - In 56 instances the RAMP generated a solution with the same quality
  - In 81 instances the average time to find the best solution was less than or equal
  - In 74 instances the average percentage deviation from the best available solution was smaller
RAMP components analysis

- $\text{RAMP}_{\text{UNIF}}$, reblocking structure replaced by uniformly random solutions
- $\text{RAMP}_{\text{SWAP}}$, discards insertion neighborhood, only swap neighborhood
- $\text{RAMP}_{\text{INSERT}}$, discards the swap neighborhood, only insertion neighborhood
RAMP components analysis

Average deviation percentage from the best known solution in the literature

- RAMP_UNIF
- RAMP_SWAP
- RAMP_INSERT
- RAMP
RAMP components analysis
Reblocking and uniform construction analysis

Estimated probability density functions for reblocking and uniform construction

Runtime of reblocking and uniform algorithms
Conclusion

- RAMP performed comparatively to NEGA_VNS.
- RAMP outperformed its three variants, RAMP_UNIF, RAMP_SWAP and RAMP_INSERT.
- Reblocking outperformed uniform construction.
Discussion

- **Good:**
  - Compared RAMP to 7 different algorithms
  - Compared 4 different variants of RAMP
  - Extensive comparison with NEGA\textsubscript{VNS}, a competitive algorithm
  - CPU scaling factor and the issue of different architectures are acknowledged

- **Less good:**
  - Table 5 shows NEGA\textsubscript{VNS} performs better than RAMP for larger problems, this is not mentioned
  - Scaling factor only incorporates CPU clock speed
    - CPU architecture or compiler version not mentioned, ipc and compiler optimization unclear
  - HGA_RMA is suddenly excluded after table 3
  - Small inconsistencies in the paper (eg. RAMP\textsubscript{INSERTION} vs RAMP\textsubscript{INSERT}, re-blocking vs reblocking)
An Effective Evolutionary Hybrid for Solving the Permutation Flowshop Scheduling Problem

Amirghasemi & Zamani (2015)

Finn van der Heide, Job Hartjes, Mick van het Nederend