Korygowanie harmonogramów z uwzględnieniem awarii maszyn

Kamil Niemczyk

Politechnika Wrocławska
Automatyka i Robotyka, Wydział Elektroniki
Technologie informacyjne w systemach automatyki

Plan prezentacji

1. Korygowanie harmonogramów z uwzględnieniem zakłóceń
   - Mathematical model
   - Cost function
2. Tabu search with neural network
   - Neighborhood
   - Tabu effect
3. Parallel neuro-tabu search
   - Diversification algorithm
   - Intensification algorithm
4. Computational experiments
5. Conclusions

Zaproponowano algorytm Tabu Search z nawrotami, który przystosowano do rozwiązywania elastycznych problemów gniazdowych.

Elastyczny problem gniazdowy

Let us consider

- $J = \{1, 2, \ldots, n\}$ – a set of jobs,
- $M = \{1, 2, \ldots, m\}$ – a set of machines,
- $O = \{1, 2, \ldots, o\}$ – a set of operations.

A job j consists of the sequence of o_j operations. An operation i has to be executed on the dedicated machine without interruptions in time $p_i > 0$, $i \in O$. The solution is a vector of times $S = (S_1, S_2, \ldots, S_o)$ of operations beginning such, that:

- a job begins its execution on the next machine if it is finished on the previous one,
- a job begins on the machine if the previous job executed on this machine is finished,
- beginning times are not negative.

The job shop problem

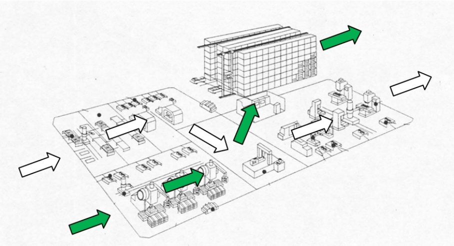
A feasible solution: a vector $S = (S_1, S_2, \ldots, S_o)$ such, that:

$$S_{l_{j-1}+1} \geq 0, \quad j = 1, 2, \ldots, n,$$

$$S_i + p_i \leq S_{i+1}, \quad i = l_{j-1}+1, \; l_{j-1}+2, \ldots, l_j - 1, \quad j = 1, 2, \ldots, n,$$

$$S_i + p_i \leq S_j \quad \text{or} \quad S_j + p_j \leq S_i, \quad i, j \in O, \quad v_i = v_j, \quad i \neq j.$$

where $C_j = S_j + p_j$, a machine number v_i of a job i.



The job shop problem

An appropriate criterion function has to be added to the problem constraints:

- minimization of the time of finishing all the jobs

$$C_{\max}(S) = \max_{1 \le j \le n} C_{l_j},$$

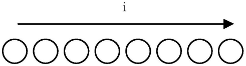
- minimization of the sum of job finishing times

$$C(S) = \sum_{j=1}^{n} C_{l_j}.$$

Both problems described are strongly NP-hard and although they are similarly modelled, the second one is found to be harder because of the lack of some specific properties (so-called block properties).

Neighborhood

In the considered neuro-tabu search algorithm *NTS* each move is represented by its neuron. For the adjacent swap neighborhood a network of neurons formed of $o-1$ neurons. Let i-th neuron represents a move consisting in swap of two adjacent elements on the positions i and $i+1$ in a solution π.



If *i* - *th* neuron fire

$\Downarrow$

$\pi : (\pi(1), \ldots, \pi(i), \pi(i+1), \ldots, \pi(o))$

$\Downarrow$

$\pi' : (\pi(1), \ldots, \pi(i+1), \pi(i), \ldots, \pi(o))$

Tabu effect

In a proposed neural network architecture a history of each neuron is stored as its internal state (*tabu effect*). Each neuron is defined by the following equations:

$$\eta_i(t+1) = \alpha \Delta_i(t),$$

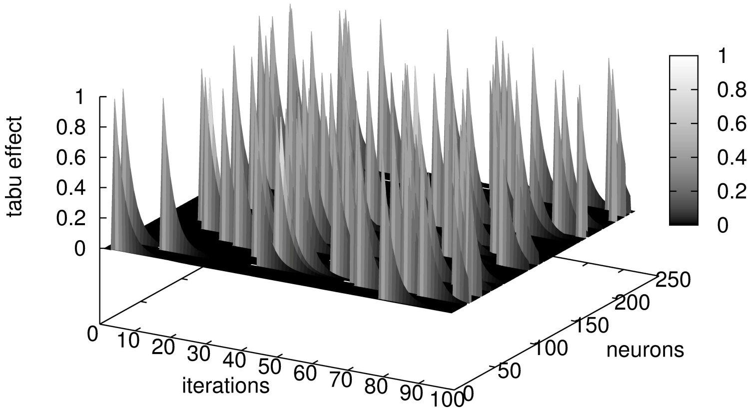
$$\Delta_i(t) = \frac{C_{max}(\pi_v^{(t)}) - C^*_{max}}{C^*_{max}},$$

$$\gamma_i(t+1) = \sum_{d=0}^{s-1} k^d x_i(t-d),$$

where $x_i(t)$ is an output of the neuron i in the iteration t.

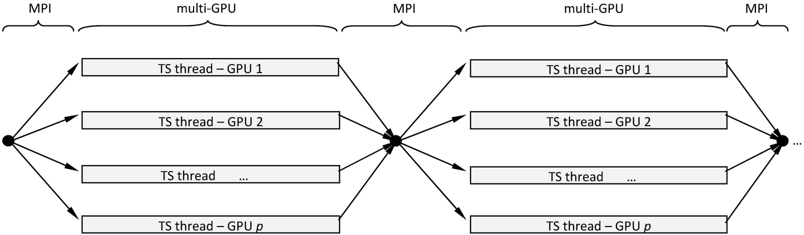
Tabu effect

A neuron i is activated if it has the lowest $(\eta_i(t+1) + \gamma_i(t+1))$ value of all the neurons.



Parallel neuro-tabu search

Here we propose a solution method to the job shop problem in the distributed computing environments, such as multi-GPU clusters. Neuro-tabu search algorithm is executed in concurrent working threads.



```
Algorithm 1.   NIS(γ, δ, C^R)
Input:  γ, δ – two processing orders;  C^R – reference makespan;
Output:  φ – processing order; update reference makespan  C^R;
  π ← γ;  iter ← 0.   Find  δ^{-1}  and  D(γ, δ)
  repeat
      iter ← iter + 1;  Find  N(π);
      For any  v ∈ N(π)  calculate and store  C_max(π_(v));
      Find  N^+ = {v = (x, y) ∈ N(π) : δ^{-1}(y) < δ^{-1}(x)};
      if  N^+ ≠ ∅  than  K ← N^+  else  K ← N(π);
      Select the move  w ∈ K  such, that
            C_max(π_(w)) = min_{v∈K} C_max(π_(v));
      Denote  π_(w)  by  α;  π ← α;  φ ← π;
      if  C_max(π) < C^R  than  C^R ← C_max(π)  and exit;
  until  iter ≥ maxV · D(γ, δ);  {maxV ∈ (0, 1) - parameter}
```

Algorithm 2. *iNTS*

Input: π^0 – processing orders provided by INSA;

Output: π^* – the best found processing order and its makespan C^*;

$(\pi^1, C^1) \leftarrow NTS(\pi^0)$; $C^* \leftarrow C^1$;

for $i \leftarrow 2, \ldots, maxE$ do

$\varphi \leftarrow NIS(\pi^{i-1}, \pi^0, C^*)$; $(\pi^i, C^i) \leftarrow NTS(\varphi)$;

$C^* = \min\{C^*, C^i\}$;

repeat

Find $1 \leq l \leq maxE$ so that

$D(\pi^k, \pi^l) = \max\{D(\pi^k, \pi^i) : 1 \leq i \leq maxE\}$;

$\varphi \leftarrow NIS(\pi^k, \pi^l, C^*)$; $(\pi^l, C^l) \leftarrow \textbf{NeuroTS}(\varphi)$;

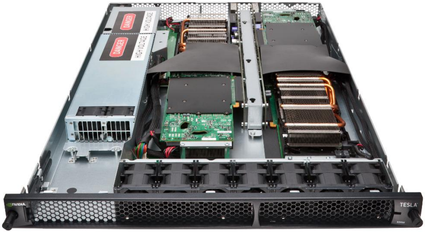
if $C^l < C^k$ than set $(\pi^*, C^*) \leftarrow (\pi^l, C^l)$ and $k \leftarrow l$;

until $\max\{D(\pi^k, \pi^i) : 1 \leq i \leq maxE\} < maxD$.

Computational experiments

Proposed algorithms were ran on the server based on 6-cores Intel Core i7 CPU X980 (3.33GHz) processor equipped with nVidia Tesla S2050 GPU (1792 cores).

- *sNTS* – sequential neuro-tabu Search algorithm,
- *pNTS* – parallel (for $p = 16$) neuro-tabu Search algorithm,
- *iNTS* – advanced *NTS* algorithm based on the diversification and intensification methodology.



Results of experiments

problem	$n \times m$	$sNTS$	$pNTS(p = 16)$	$iNTS$
TA01-10	15×15	0.4948	0.3141	0.0652
TA11-20	20×15	1.1691	0.9129	0.4412
TA21-30	20×20	1.2486	0.7033	0.4803
TA31-40	30×15	1.0592	0.7965	0.3764
TA41-50	30×20	1.8565	1.5634	0.8328
TA51-60	50×15	0.0915	0.0915	0.0520
TA61-70	50×20	0.1479	0.0210	0.0140
TA71-80	100×20	0.0090	0.0090	0.0090
average		**0.7596**	**0.5515**	**0.2839**

Table: Percentage relative deviations (PRD) to the best known solutions.

Conclusions

- We propose an approach designed to solve difficult problems of combinatorial optimization in distributed parallel architectures without shared memory, such as clusters of nodes equipped with GPU units (i.e. multi-GPU clusters).
- The methodology can be especially effective for large instances of hard to solve optimization problems, such as flexible scheduling problems as well as discrete routing and assignment problems.