Large neighbourhood search algorithm for type-II assembly line balancing problem

Tip-II montaj hatti dengeleme problemi için büyük komşuluk arama algoritması

Şener AKPINAR

4Department of Industrial Engineering, Engineering Faculty, Dokuz Eylul University, Izmir, Turkey.

Received/Geliş Tarihi: 22.05.2016, Accepted/Kabul Tarihi: 22.08.2016

Abstract
This paper proposes a large neighbourhood search (LNS) algorithm for type-II simple assembly line balancing problem (SALBP-II). The LNS algorithm was initially developed for solving vehicle routing problem and its later implementations were used to solve scheduling problems. The reported results about these two problems indicate that LNS algorithm is a powerful method. Vehicle routing problem has the main objective to find the optimal match between a certain number of routes and a certain number of customers, while SALBP-II is trying to find the optimal match between a certain number of workstations and a certain number of assembly operations. To our point of view, LNS algorithm would also be a powerful method for solving SALBP-II due to the structural similarity between these two problems. Within this context, a LNS algorithm is developed to tackle SALBP-II and the performance of the proposed algorithm is tested on a set of benchmark instances. Computational results indicate the satisfactory performance of LNS algorithm in solving SALBP-II.

Keywords: Large neighbourhood search algorithm, Assembly line balancing, Maximizing production rate, Meta-heuristic

1 Introduction
The history of the assembly lines begins with the Terracotta Army, which was produced in an assembly type manner in China in 200 BC, however, assembly lines played their vital role in automotive industry after the development of the first automotive assembly line by Ransom Olds in 1901 [1]. In 1913, Henry Ford improved Ransom Olds’ idea by introducing the moving belt for the assembly lines of his factory, where he had been produced Model T Fords. This innovative improvement on the automobile production process changed the type of manufacturing systems all over the world over the years because of the significantly reduced cost of production through the investment and installation of assembly lines (ALs). Through these kind of production systems companies have been able to produce standardized products in high volumes. Additionally, this type of production systems picked out a new optimization problem namely assembly line balancing problem (ALBP) that mainly deals with the design issues of ALs. ALBP searches for the allocation of task to workstations under some constraints in order to optimize some pre-determined performance measures.

Salveson formulated ALBP mathematically for the first time in literature and focused on line configuration, assignment of tasks to workstations [2]. Afterwards, ALBP has been attracted a lot of attention in the research community. This interest of the research community on the ALBP has been increased continuously. The reader can be informed about the literature on ALBPs through the comprehensive review studies of [3]-[7].

This paper considers the single-model assembly lines, designed to produce high volumes of standardized products, and simple assembly line balancing problem (SALBP) [8] tries to calibrate such lines. In the relevant literature, SALBP has been tackled as SALBP-I and SALBP-II in order to give some decisions concerning the design and operational levels, respectively. SALBP-II (respectively SALBP-I) is related to assignment of assemble tasks to workstations in order to identify minimum cycle time (respectively minimum number of workstations) for a pre-determined number of workstations (respectively cycle time) while satisfying the problem constraints. Both SALBP-I and SALBP-II are NP hard [9], and SALBP-I is much more popular among the researchers than the SALBP-II [7]. The reader can be suggested to see the review paper of [8] for detailed discussions about the solution approaches of simple assembly line balancing problems.

Solution approaches for SLABP-II can be evaluated as direct and iterative solution strategies. A direct solution method aims at identifying the optimum solution directly, while an iterative solution approach tries to identify the optimum solution by tackling SALBP-I for different trial cycle times.
Additionally, solution methods on SALBP-II can be grouped as exact and heuristic approaches. The exact methods developed in [10] and [9] are able to solve the SALBP-II iteratively, while the one developed in [11] is able to solve the problem directly. Moreover, integer-programming formulations for SALBP-II were developed in [12] and [13]. Besides these exact methods, the SALBP-II literature covers many heuristic and meta-heuristic approaches. A linear programming based two-stage heuristic was proposed in [14] while a Petri nets based heuristic was proposed in [15]. Another heuristic method that starts an initial solution and realizes trade, and transfer procedures to improve this solution was proposed in [16]. Genetic algorithms [17]-[19], tabu search procedures [20],[21], simulated annealing algorithms [22],[23], ant colony optimization algorithms [24],[25], particle swarm optimization algorithm [26], differential evolution algorithm [27],[28], beam search algorithm [28]-[30] and variable neighbourhood search [31],[32] were also used to solve SALBP-II. In this paper, we propose a large neighbourhood search (LNS) algorithm for solving the SALBP-II. The LNS algorithm and its variants have been successfully implemented to solve vehicle routing problems in literature (see Section 2). Vehicle routing problem and the SALBP-II have structural similarities as the vehicle routing problem is trying to optimally match a certain number of routes and a certain number of customers, while SALBP-II is trying to optimally match a certain number of workstations and a certain number of assembly operations. Therefore, LNS algorithm would also be a powerful method for solving SALBP-II due to the structural similarity between these two problems. To the best of our knowledge, this is the first attempt to solve the SALBP-II via the LNS algorithm, however a simulation-based adaptive large neighborhood search heuristic was developed in [33] for a specific case of assembly line balancing problem, designing of a footwear assembly line under stochastic task time and parallel workstations.

The remainder of this paper is organized as follows. Basic steps of the LNS algorithm are given in Section 2. The proposed LNS algorithm for solving SALBP-II is presented in Section 3. Computational study is presented in Section 4. Conclusions are given in Section 5.

2 Large neighbourhood search algorithm

**Nomenclature**

- $s$ : A feasible solution,
- $s'$ : Solution under improvement,
- $s'_{best}$ : Best solution,
- $f(s')$ : Fitness value of the solution $s'$,
- $f(s'_{best})$ : Fitness value of the best solution,
- $q$ and $p$ : User defined parameters,
- $L$ : An array sorting individual effects of tasks on a solution,
- $y$ : A random number from the interval $[0,1]$,
- $r$ : Task to be removed from a solution

Shaw [34] initially proposed the LNS algorithm for solving vehicle routing problem with time windows (VRPTW). The LNS algorithm tries to explore the solution space of an optimization problem through two successive operators: destruction of a solution by a removal heuristic and reparation of the destroyed solution by an insertion heuristic. A removal heuristic generates an infeasible solution by removing some components of a solution, while an insertion heuristic turns this infeasible solution to a feasible solution by reinserting the removed components with the guidance of a rule. Then, LNS algorithm accepts or declines this solution via an acceptance function. LNS algorithm executes these steps successively as can be seen from Figure 1 until the stopping condition is met.

```
Initialize $s \in \text{solution}$s and $q \in \mathbb{N}$

$s'_{best} \leftarrow s$

repeat

$s' \leftarrow s$

project $q$ components out of $s'$

reinsert removed components into $s'$

if $(f(s') < f(s'_{best}))$ then

$s'_{best} \leftarrow s'$

if accept($s', s$) then

$s \leftarrow s'$

until termination condition met

report $s'_{best}$
```

Figure 1: Main steps of the LNS algorithm [35].

LNS algorithm and its variants are powerful methods for routing and scheduling problems in general [36]. Ropke and Pisinger modified the basic LNS algorithm and they named the modified algorithm as adaptive large neighbourhood search (ALNS) algorithm [35]. ALNS has many successful implementations in literature [37]-[44]. Within the scope of this current paper, we have the aim of tackling the SALBP-II via a LNS algorithm and this attempt will be the first one for solving an ALBP via a LNS algorithm in literature as far as we know.

3 A large neighbourhood search algorithm for SALBP-II

This section introduces the proposed LNS algorithm for solving SALBP-II in depth. The proposed algorithm initializes itself by randomly generating a feasible solution and tries to improve this solution through removal and insertion heuristics. The algorithm terminates itself when the stopping condition is achieved. The following sub-sections introduce the main steps of the proposed LNS algorithm for SALBP-II.

3.1 Solution representation

In the proposed LNS algorithm, we used task based representation [45],[46] which is used for type-I ALBPs in general. The number of tasks defines the length of the representation schema; however, the situation is different for type-II problems. A solution representation schema of type-II ALBPs must contain the information about the pre-determined number of workstations. Because of this reason, the original task based representation must be modified to code a solution of the SALBP-II. Within this context, we modified the original task based representation as illustrated in Figure 2.
3.2 Initial solution

As mentioned before, the proposed LNS algorithm starts with a randomly generated initial feasible solution and tries to improve this solution through its operators. For the SALBP-II, the initial solution generation mechanism firstly determines a task sequence that satisfies the precedence relations between tasks. After that, it divides the related task sequence into $s$ number of workstations by inserting $s−1$ separators into randomly selected positions within the task sequence. Figure 3 illustrates the initial solution generation mechanism for an assembly line having 11 tasks and 3 workstations.

![Initial solution generation mechanism](image)

The precedence relations and the task performing times are shown in Figure 3a and the Figure 3b illustrates a feasible task sequence according to precedence relations among tasks. For determining tasks assignments to three workstations 2 separators are inserted into the randomly determined positions within the feasible task sequence as can be seen from Figure 3c. According to the randomly generated solution in Figure 3c, tasks 1, 4, 8 and 5 are performed in workstation 1, tasks 3, 2 and 9 are performed in workstation 2, and tasks 6, 10, 7 and 11 are performed in workstation 3. These assignments of tasks result to workloads for workstations 1, 2 and 3 as 20, 10 and 15, separately. As a result, the cycle time of the assembly line is determined as 20 due to the solution depicted in Figure 3.

3.3 Removal heuristic

The proposed LNS algorithm realizes a removal heuristic based on the worst removal heuristic [35] in order to destroy the current solution. This main idea of this heuristic is specifying the misplaced tasks within a solution and inserting them into another position in the current solution. Within this scope, the proposed removal heuristic calculates a cost value of performing task $i$ in its current position in the current solution as $\text{cost}(i, s) = f(s) − f_i(s)$ where $f(s)$ is the cost of the complete solution and $f_i(s)$ is the cost of the solution without task $i$. For the SALBP-II implementation of the worst removal heuristic, we considered the total idle time occurring on the assembly line. Hereby, the proposed removal heuristic aims at identifying the tasks causing high idle times within the assembly line according to current solution. In other words, the proposed removal heuristic aims at specifying the tasks inducing high cycle time in its current position in the current solution. The algorithm of the worst removal heuristic is depicted in Figure 4.

![Worst removal heuristic](image)

As can be seen from Figure 4, removal heuristic uses two parameters as $q$ and $p$ for determining the number of tasks to be removed and avoiding repeatedly removing same tasks, respectively. As mentioned before, this heuristic tries to specify the misplaced tasks in a solution and therefore identifies the tasks required to move to another position in the current solution. Various removal heuristics were developed especially for the adaptive versions of the LNS algorithm, but there is no information about the best removal heuristic in literature. Therefore, we select the more reasonable removal heuristic for us, since our focus is to implement the LNS algorithm for SALBP-II not identifying the best removal heuristic within the context of this paper.

3.4 Insertion heuristic

In this paper, we used an insertion heuristic based on the ranked positional weight technique (RPWT) [47], which calculates positional weights for all the tasks of an assembly line according to the precedence relations and task performing times. The positional weights are then used as priority values while assigning tasks to workstations. For the SALBP-II implementation of the LNS algorithm, it is required to determine the order of reinserting the removed tasks by the
removal heuristic. Within this scope, the insertion heuristic used in this paper firstly ranks the removed tasks according to their positional weights in descending order. After that, the insertion heuristic tries to reinsert the removed tasks in its best position in accordance with their ranks. Various insertion heuristics because of different priority values can be used for the SALBP-II implementation of the LNS algorithm, however we mainly focus on the implementation of LNS algorithm to SALBP-II not selecting the best insertion heuristic within the context of this study.

3.5 Solution acceptance and stopping criteria

The original LNS algorithm accepts only the improving solutions; however, this may cause to be trapped in a local optimum. Therefore, the later variants of LNS algorithm realize an acceptance function in order to specify the current solution. In our implementation of LNS algorithm to SALBP-II, we use an acceptance criteria based on the simulated annealing algorithm’s acceptance criteria as realized in [35]. That is, the proposed LNS algorithm accepts a solution sol’ in comparison to the current solution sol with the probability of 

\[ e^{-\left(f(sol')-f(sol)/T\right)} \]

where \( T > 0 \) defines the current iteration’s temperature. The temperature has an initial value as \( T_{\text{start}} \) and it decreases via the rule \( T = T \cdot cr \), where \( 0 < cr < 1 \) is the cooling ratio. The proposed LNS algorithm terminates itself after executing a pre-determined number of iterations or achieving the previously known optimal solution of a problem.

4 Computational study

This section evaluates the performance of the proposed LNS algorithm. For this purpose, we used the benchmark data sets downloaded from the website of http://alb.mansci.de/. The benchmark sets contain 302 instances in total. 128 instances based on 9 different precedence graphs belong to the data set 1 with the number of tasks varying from 29 to 111 and the rest of the 174 instances based on 8 different precedence graphs belong to the data set 2 with the number of tasks varying from 53 to 297. Each SALBP-II instance is defined with the precedence relations between tasks illustrated via a precedence graph \( G \) and the processing times of each task. Moreover, \( n \) number of tasks must be assigned to a pre-determined number of \( m \) workstations in order to identify the existing minimal cycle time \( c^\ast \).

4.1 Results and Discussion

In this sub-section, we have directly compared the proposed LNS algorithm to 15 formerly developed SALBP-II methods, 3 versions of genetic algorithm (pGA [48], rGA_prb and rGA_rks [49]), 4 versions of differential evaluation algorithm (DE_prb and DE_rks [27], IDEA_rk2 and IDEA_opt [28]), 3 petri net based heuristics (PNA_for, PNA_back and PNA_bid [15]), 4 versions of particle swarm optimization (PSO1, PSO2, PSO3 and PSO4 [26]) and iterated beam search (IBS [29]). The proposed LNS algorithm was coded in Matlab 7.9 and then the results were obtained by running the coded LNS on a Core(TM) i7-2640 CPU 2.80 GHz personal computer. Additionally, LNS requires tuning some parameters as summarized in Table 1 with their values and definitions.

The values of the parameters were determined through a preliminary experimental study executed on the problem of Killbridge with 45 tasks and 11 workstations. This preliminary study was performed with three levels of MaxIter \( \in \{50,100,150\} \), three levels of the lower limit (LL) and upper limit (UL) for the rule that randomly selects the number of tasks to be removed as \( LL \in \{4,8,12\} \) and \( UL \in \{\min(50,0.2n), \min(50,0.4n), \min(50,0.6n)\} \), and three levels \( p \in \{4,6,8\} \). The values for the parameters \( T_{\text{start}} \) and \( cr \) were taken from [50]. This experimental study done for parameter tuning resulted with the parameter values given in Table 1 by considering the quality of the generated solutions and computational time spent by the algorithm.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxIter</td>
<td>Maximum number of iterations</td>
<td>100*n</td>
</tr>
<tr>
<td>q</td>
<td>Number of tasks to be removed</td>
<td>4 ≤ randi(4) ≤ min(50,0.4n)</td>
</tr>
<tr>
<td>p</td>
<td>Parameter used to avoid repeatedly selecting same task for removal</td>
<td>6</td>
</tr>
<tr>
<td>T_{\text{start}}</td>
<td>Initial temperature</td>
<td>100</td>
</tr>
<tr>
<td>cr</td>
<td>Cooling ratio</td>
<td>0.99975</td>
</tr>
</tbody>
</table>

Through this preliminary study we can provide following observations. The higher values of MaxIter caused redundant iterations while its smaller values result with disappointed performance of the algorithm. Higher values of the LL of the random selection rule caused unsatisfactorily level of intensification, while the lower values of the UL of the random selection rule caused unsatisfactorily level of diversification. Finally, lower values of \( p \) coresponds to much randomness as stated in [35]. The results obtained via the proposed LNS algorithm and the provided results of the other 15 algorithms on the benchmark sets are reported in Table 2. We ran the proposed LNS algorithm 10 times for all the test problems, since the reported results for the majority of other methods had also been obtained through 10 independent runs of algorithms for each problem. Table 2 provides the following information for all the methods.

- \( \text{c}\%\text{dev} \): The average relative deviation from the known optimum solution (previously known minimal cycle time \( c^\ast \)) in percentage; calculated as \((c - c^\ast)/c^\ast\)100; where \( c \) is the best cycle time achieved by the solution algorithm.

From the observations of Table 2, it can be clearly seen that the proposed LNS algorithm outperformed the variants of genetic algorithm, differential evaluation algorithm, particle swarm optimization and petri net based heuristics with respect to the c%dev achievements, which is related to the main optimization criteria of cycle time. However, IBS outperforms all the other algorithms, as well as LNS as can be obviously seen from Table 2. The results of all the algorithms are visualized in Figure 5, which displays the fluctuations of c%dev values over the benchmark problems, in order to emphasize the effective performance of the proposed LNS algorithm in solving SALBP-II instances.

The effectiveness of the LNS algorithm in solving the SALBP-II instances can be clearly seen if the average %dev values provided for the data sets seperately taken into consideration.
LNS was able to generate solutions for data set 1 and data set 2 with an average %dev value of 0.78 and 1.02, respectively. Therefore, we can conclude that LNS is a powerful method for solving SALBP-II instances. Finally, average %dev values in comparison to other methods are visualized in Figure 6 in order to provide a better understanding.

5 Conclusions
The main idea of this study was to tackle SALBP-II with a LNS algorithm for the first time in literature as far as we know. The original LNS algorithm and its later variants were generally used to tackle vehicle routing and scheduling problems. Especially for the vehicle routing problems, variants of the LNS algorithm is able to produce satisfactory results. Due to the similar nature of vehicle routing, and type-II ALBP, it was thought that the LNS algorithm would be a satisfactory method in solving type-II ALBPs. Within this context, a LNS algorithm was developed to tackle a well-known assembly line balancing problem of SALBP-II.

Table 2: Computational results over the benchmark set.

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>n</th>
<th>pGA</th>
<th>rGA_rks</th>
<th>prb</th>
<th>DE_rks</th>
<th>IDEA_rk</th>
<th>IDEA_rk2</th>
<th>IDEA_apt</th>
<th>PNA_for</th>
<th>PNA_back</th>
<th>PNA_bid</th>
<th>PSO1</th>
<th>PSO2</th>
<th>PSO3</th>
<th>PSO4</th>
<th>IBS</th>
<th>LNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Busey</td>
<td>29</td>
<td>3.26</td>
<td>2.15</td>
<td>2.46</td>
<td>2.69</td>
<td>1.16</td>
<td>2.21</td>
<td>3.07</td>
<td>3.07</td>
<td>2.92</td>
<td>3.76</td>
<td>1.96</td>
<td>5.88</td>
<td>1.55</td>
<td>5.01</td>
<td>0</td>
<td>0.55</td>
</tr>
<tr>
<td>Sawyer</td>
<td>30</td>
<td>2.61</td>
<td>4.51</td>
<td>3.67</td>
<td>3.52</td>
<td>2.27</td>
<td>3.34</td>
<td>3.52</td>
<td>4.00</td>
<td>4.39</td>
<td>5.24</td>
<td>2.24</td>
<td>6.62</td>
<td>1.60</td>
<td>5.61</td>
<td>0</td>
<td>1.22</td>
</tr>
<tr>
<td>Lutz1</td>
<td>32</td>
<td>0.32</td>
<td>0.71</td>
<td>0</td>
<td>0.35</td>
<td>0.32</td>
<td>0.35</td>
<td>0.71</td>
<td>4.09</td>
<td>2.93</td>
<td>4.09</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0.32</td>
</tr>
<tr>
<td>Gunther</td>
<td>35</td>
<td>0.64</td>
<td>1.71</td>
<td>1.21</td>
<td>1.02</td>
<td>0.14</td>
<td>1.02</td>
<td>1.27</td>
<td>1.27</td>
<td>2.99</td>
<td>1.27</td>
<td>0.87</td>
<td>3.88</td>
<td>0.64</td>
<td>3.92</td>
<td>0</td>
<td>0.42</td>
</tr>
<tr>
<td>Kilbridge</td>
<td>45</td>
<td>1.38</td>
<td>0.96</td>
<td>0.8</td>
<td>0.6</td>
<td>0.66</td>
<td>0.60</td>
<td>0.60</td>
<td>2.19</td>
<td>1.46</td>
<td>1.91</td>
<td>0.80</td>
<td>1.64</td>
<td>0.68</td>
<td>1.78</td>
<td>0</td>
<td>0.07</td>
</tr>
<tr>
<td>Tonge</td>
<td>70</td>
<td>2.75</td>
<td>3.63</td>
<td>3.63</td>
<td>2.07</td>
<td>1.88</td>
<td>2.07</td>
<td>1.78</td>
<td>2.53</td>
<td>3.41</td>
<td>2.92</td>
<td>2.91</td>
<td>5.99</td>
<td>2.03</td>
<td>5.82</td>
<td>0</td>
<td>1.03</td>
</tr>
<tr>
<td>Arcus1</td>
<td>83</td>
<td>1.65</td>
<td>1.96</td>
<td>1.48</td>
<td>0.83</td>
<td>0.99</td>
<td>0.98</td>
<td>0.78</td>
<td>2.47</td>
<td>2.57</td>
<td>2.25</td>
<td>1.42</td>
<td>3.20</td>
<td>0.70</td>
<td>3.04</td>
<td>0.0287</td>
<td>0.70</td>
</tr>
<tr>
<td>Lutz2</td>
<td>89</td>
<td>3.13</td>
<td>3.76</td>
<td>2.84</td>
<td>2.54</td>
<td>3.08</td>
<td>2.64</td>
<td>1.99</td>
<td>2.99</td>
<td>4.44</td>
<td>3.92</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0.85</td>
</tr>
<tr>
<td>Arcus2</td>
<td>111</td>
<td>5.02</td>
<td>5.27</td>
<td>5.11</td>
<td>4.98</td>
<td>4.96</td>
<td>4.98</td>
<td>4.64</td>
<td>2.06</td>
<td>4.61</td>
<td>2.25</td>
<td>3.79</td>
<td>6.67</td>
<td>2.02</td>
<td>5.77</td>
<td>0.0066</td>
<td>1.82</td>
</tr>
<tr>
<td>Average</td>
<td>2.31</td>
<td>2.74</td>
<td>2.36</td>
<td>2.07</td>
<td>1.72</td>
<td>2.02</td>
<td>2.04</td>
<td>2.57</td>
<td>3.51</td>
<td>2.88</td>
<td>2.00</td>
<td>4.84</td>
<td>1.32</td>
<td>4.42</td>
<td>0.0058</td>
<td>0.78</td>
<td></td>
</tr>
<tr>
<td>Data Set 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hahn</td>
<td>53</td>
<td>1.27</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.52</td>
<td>2.08</td>
<td>2.61</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0.50</td>
</tr>
<tr>
<td>Warnecke</td>
<td>58</td>
<td>4.29</td>
<td>6.40</td>
<td>5.25</td>
<td>3.31</td>
<td>3.74</td>
<td>3.98</td>
<td>3.51</td>
<td>3.51</td>
<td>5.57</td>
<td>5.90</td>
<td>6.01</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.0579</td>
<td>0.35</td>
</tr>
<tr>
<td>Wee-Mag</td>
<td>75</td>
<td>2.61</td>
<td>2.37</td>
<td>2.44</td>
<td>1.39</td>
<td>1.23</td>
<td>1.63</td>
<td>1.39</td>
<td>1.56</td>
<td>1.63</td>
<td>1.57</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lutz3</td>
<td>89</td>
<td>2.54</td>
<td>3.20</td>
<td>2.19</td>
<td>1.58</td>
<td>1.68</td>
<td>2.88</td>
<td>1.57</td>
<td>2.59</td>
<td>2.88</td>
<td>2.94</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0.78</td>
</tr>
<tr>
<td>Mukherje</td>
<td>94</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.04</td>
<td>1.41</td>
<td>1.21</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0.64</td>
</tr>
<tr>
<td>Barthold1</td>
<td>148</td>
<td>0.85</td>
<td>0.76</td>
<td>0.58</td>
<td>0.24</td>
<td>0.26</td>
<td>0.46</td>
<td>0.26</td>
<td>1.02</td>
<td>0.46</td>
<td>0.48</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0.64</td>
</tr>
<tr>
<td>Barthold2</td>
<td>148</td>
<td>9.64</td>
<td>9.77</td>
<td>9.88</td>
<td>7.37</td>
<td>6.85</td>
<td>4.79</td>
<td>4.79</td>
<td>3.97</td>
<td>4.79</td>
<td>4.32</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>2.14</td>
</tr>
<tr>
<td>Scholl</td>
<td>297</td>
<td>10.16</td>
<td>11.76</td>
<td>10.31</td>
<td>9.87</td>
<td>9.51</td>
<td>10.16</td>
<td>9.24</td>
<td>3.21</td>
<td>2.90</td>
<td>2.75</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.0028</td>
<td>3.13</td>
</tr>
<tr>
<td>Average</td>
<td>4.27</td>
<td>5.08</td>
<td>4.38</td>
<td>3.39</td>
<td>3.32</td>
<td>3.41</td>
<td>2.97</td>
<td>2.85</td>
<td>3.01</td>
<td>2.93</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.00071</td>
<td>1.02</td>
</tr>
</tbody>
</table>

Figure 5: The average relative deviations over the benchmark instances.
The performance of the proposed LNS algorithm was evaluated on a set of SALBP-II instances against 15 formerly developed SALBP-II methods, 3 versions of genetic algorithm, 4 versions of differential evaluation algorithm, 3 petri net based heuristics, 4 versions of particle swarm optimization and iterated beam search. Computational results demonstrate that the proposed LNS algorithm is capable to SALBP-II instances with a satisfactory performance.

Future researches will focus on tackling the different versions of ALBP via the proposed LNS algorithm and developing some other variants of the LNS algorithm for the ALBPs in order to improve the effectiveness of the proposed LNS algorithm.

6 References


