On Solving the Assembly Line Worker Assignment and Balancing Problem via Beam Search[☆]

Christian Blum[*,a], Cristobal Miralles[b]

[a]*ALBCOM Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain*
[b]*ROGLE – Dpto. Organización de Empresas, Universidad Politécnica de Valencia, Valencia, Spain*

Abstract

Certain types of manufacturing processes can be modelled by assembly line balancing problems. In this work we deal with a specific assembly line balancing problem that is know as the assembly line worker assignment and balancing problem (ALWABP). This problem appears in settings where tasks must be assigned to workers, and workers to work stations. Task processing times are worker specific, and workers might even be incompatible with certain tasks. The ALWABP was introduced to model assembly lines typical for sheltered work centers for the Disabled.

In this paper we introduce an algorithm based on beam search for solving the ALWABP with the objective of minimizing the cycle time when given a fixed number of work stations, respectively workers. The experimental results show that our algorithm is currently a state-of-the-art method for this version of the ALWABP. In comparison to results from the literature, our algorithm obtains better or equal results in all cases. Moreover, the algorithm is very robust for what concerns the application to problem instances of different characteristics.

Key words:
beam search, assembly line worker assignment and balancing

1. Introduction

Assembly line balancing (ALB) [1] concerns the optimization of processes related to the manufacturing of products via assembly lines. The specific problem considered in this paper is a generalization of the so-called simple assembly line balancing problem (SALBP) [2], which is a well-studied scientific test case.

[☆]This work was supported by grant TIN2007-66523 (FORMALISM) of the Spanish government. Moreover, Christian Blum acknowledges support from the *Ramón y Cajal* program of the Spanish Ministry of Science and Innovation.

[*]Corresponding author

Email addresses: cblum@lsi.upc.edu (Christian Blum), cmiralles@omp.upv.es (Cristobal Miralles)

In SALBP, an assembly line consists of a set of work stations arranged in a straight line, and by a transport system which moves the product to be manufactured along the line. The product is manufactured by performing a given set of tasks, each of which has a processing time. A solution to a SALBP instance is obtained by the assignment of all tasks to work stations subject to precedence constraints between the tasks. The fact that all work stations are equally sized and the assembly line moves in constant speed, implies a maximum of C time units (the cycle time) for processing all the tasks assigned to a work station. Among several possible goals for optimization, the following two are the ones that were most studied in the literature. Given a fixed cycle time C, the optimization goal consists in minimizing the number of necessary work stations. This problem version is called SALBP-1 in the literature. Given a fixed number m of work stations, the goal is to minimize the cycle time C. The literature knows this second problem version as SALBP-2.

In this paper we consider a generalization of SALBP-2: the so-called assembly line worker assignment and balancing problem (ALWABP). This problem was introduced in [3], motivated by the growing need and desire to incorporate the Disabled into the active workforce. The World Health Organization estimates that 10% of the global population, which amounts to around 610 million people worldwide, is disabled. Of these, 386 million people are within the active labor age range, but experience very high unemployment rates (fluctuating from 13% in the UK to 80% in many under-developed countries). This has led to various attempts of integrating these citizens into the working society. Indeed, under the concept of Corporate Social Responsibility (see, for example, [4]), an increasing number of companies are becoming concerned with this matter. In this context, the employment of disabled workers is seen as a way of including the interests of society in the company goals.

One of the strategies most commonly adopted for facilitating the integration of disabled workers into the labor market is the creation of sheltered work centers for Disabled (henceforth SWDs). This model of socio-labor integration tries to move away from the traditional stereotype that considers disabled people as unable to develop continuous professional work. Just as in any other firm, a SWD competes in real markets and must be flexible and efficient enough to adapt to market fluctuations and changes, the only difference being that the SWD is a Not-For-Profit organization. Thus, the potential benefits that may be obtained from increased efficiency are usually invested into the growth of the SWD. This results in more jobs for the Disabled and the gradual integration of people with higher levels of disability, which are in fact the primary aims of every SWD.

Miralles et al. [3] have shown how the adoptation of assembly lines provides various advantages for these centers. The traditional division of work into single tasks can become a perfect tool for making certain worker disabilities invisible. In fact, an appropriate task assignment can even become a good therapeutic method for the rehabilitation of certain disabilities. However, the employed balancing procedure should be able to cope with some specific constraints relative to time variability that arise in this environment. Moreover, it should be able to reconcile the following objectives (that should no longer be seen as contradictory but complementary): (1) to maximize the efficiency of the line by balancing the workload assigned to each available worker in each station; (2) to satisfy and respect the existent constraints in this environment due the human factors when assigning tasks to workers.

After analyzing several SWDs, the authors of [3] proposed the ALWABP, which is obtained from the SALBP-2 as follows. Instead of only considering the assignment of tasks to work stations, Miralles et al. introduced in addition a set of workers that execute the tasks and that have to be assigned to work stations. Moreover, processing times of tasks are made worker-dependent. In other words, in an attempt of modelling the disabilities of different workers, each task has a worker-dependent processing time. The first mathematical model of this problem was proposed in [5]. Note that a technical description of the ALWABP is given in Section 2 below.

1.1. Previous Work

In [5], Miralles et al. presented a branch and bound algorithm for solving the ALWABP to optimality. They tested their algorithm on a range of benchmark instances. Unfortunately, only small instances (in terms of the number of tasks) could be solved. Additionally, the authors of [5] developed a simple constructive one-pass heuristic for obtaining approximate solutions to larger problem instances. The currently best performing algorithm for the ALWABP is a hybrid algorithm proposed by Chaves et al. in [6], where the authors hybridized a clustering search approach proposed in [7] with iterated local search.

1.2. Contribution of this Work

In this work we present an algorithm based on beam search for solving the ALWABP. Beam search is a classical tree search method that was introduced in the context of scheduling [8]. The central idea behind beam search is the parallel and non-independent construction of a limited number of solutions with the help of a greedy function and a lower bound to evaluate partial solutions. Our choice of beam search was motivated by the fact that the crucial algorithmic component of one of the current state-of-the-art methods for the SALBP (a special case of the ALWABP) is strongly based on beam search [9]. By the application to a wide range of benchmark instances we show that our algorithm is currently a state-of-the-art method for the ALWABP.

1.3. Paper Outline

In Section 2 we present a technical definition of the ALWABP. In Section 3 we outline the proposed algorithm. Finally, in Section 4 we present the computational results, and in Section 5 we offer conclusions and an outlook on future work.

2. The ALWABP

A technical description of the ALWABP can be given as follows. An instance (T, S, W, G) of the ALWABP consists of four components. $T = \{1, \ldots, n\}$ is a set of n tasks that must be processed by workers assigned to work stations. $S = \{1, \ldots, m\}$ is an ordered line of m work stations, where 1 is the index of the first work station and m is the index of the last one. $W = \{1, \ldots, m\}$ is a set of m workers. Each worker must be assigned to exactly one work station such that each work station is occupied by only one worker. Each task $i \in T$ has a worker-specific processing time. More specifically, for each tuple (i, h) where $i \in T$ and $h \in W$ a processing time $t_{ih} > 0$ is given. If $t_{ih} = \infty$, worker h is incompatible with task i. Finally, given is a precedence graph $G = (T, A)$, which is a directed graph without cycles whose nodes are the tasks. An arc $l_{i,j} \in A$ indicates that task i must be processed before task j. Given a task $j \in T$, we denote by $P_j \subset T$ the set of tasks that must be processed before j.

A solution is obtained by assigning workers to work stations and tasks to workers such that the precedence constraints between the tasks are satisfied. The objective function is to minimize the so-called cycle time. This problem can be expressed in the following way as an integer programming (IP) problem (see also [5]).

$$\textbf{min } z \tag{1}$$

subject to:

$$\begin{aligned}
\sum_{h \in W} \sum_{s \in S} x_{ihs} &= 1 \quad \forall i \in T && (2)\\
\sum_{s \in S} y_{hs} &\leq 1 \quad \forall h \in W && (3)\\
\sum_{h \in W} y_{hs} &\leq 1 \quad \forall s \in S && (4)\\
\sum_{h \in W} \sum_{s \in S} s x_{ihs} &\leq \sum_{h \in W} \sum_{s \in S} s x_{jhs} \quad \forall j \in T, i \in P_j && (5)\\
\sum_{i \in T} t_{ih} x_{ihs} &\leq z \quad \forall h \in W, s \in S && (6)\\
\sum_{i \in T} x_{ihs} &\leq M y_{hs} \quad \forall h \in W, s \in S && (7)\\
y_{hs} &\in \{0, 1\} \quad \forall h \in W, s \in S && (8)\\
x_{ihs} &\in \{0, 1\} \quad \forall i \in T, h \in W, s \in S && (9)\\
z &> 0 && (10)
\end{aligned}$$

This model makes use of the following variables and constants: y_{hs} is a binary variable which is set to 1 if and only if worker $h \in W$ is assigned to work station $s \in S$. Moreover, x_{ihs} is a binary variable which is set to 1 if and only if task $i \in T$ is assigned to worker $h \in W$, which in turn is assigned to work station $s \in S$. Finally, M is a sufficiently large constant which should have a value larger than $|T| \cdot |W| \cdot \max_{i \in T, h \in W}\{t_{ih} \mid t_{ih} < \infty\}$. The objective function (1) minimizes the cycle time z.[1] The constraints (2) ensure that every task $i \in T$ is assigned to a single worker $h \in W$, respectively a single work station $s \in S$. The constraints (3) and (4) ensure that each worker can only be assigned to one work station, and that in each work station there is only one worker. Constraints (5) reflect the precedence relationships between the tasks. More specifically, in case a task $j \in T$ is assigned to a worker $h \in W$, which in turn is assigned to a work station $s \in S$, all tasks $i \in P_j$ can only be assigned to workers that are assigned to work stations $s' \in S$ with $s' \leq s$. The constraints (6) and (7) ensure that the sum of the worker-specific processing times of the tasks assigned to a worker $h \in W$ do not exceed the cycle time z. As both z and y_{hs} are variables, constraints (6) and (7) are defined separately in order to maintain the linearity of the model.

2.1. Solutions to the Problem

The following representation of solutions is used for the description of the algorithm in Section 3. A solution $\tau = (\pi, \mathcal{S})$ is a tuple consisting of a permutation π of length m that stores the assignment of workers to work stations, and an ordered list $\mathcal{S} = \langle S_1, \ldots, S_m \rangle$ of m sets of tasks, where S_i denotes the set of tasks that are assigned to the worker assigned to work station i. In an abuse of notation we will sometimes call S_i a work station. More in detail, π_i denotes the worker that is assigned to work station i, and for a solution $\tau = (\pi, \mathcal{S})$ to be valid it must fulfill the following conditions:

1. $\bigcup_{i=1}^{m} S_i = \{1, \ldots, n\}$ and $\bigcap_{i=1}^{m} S_i = \emptyset$. These conditions ensure that each task is assigned to exactly one worker.
2. For each task $j \in S_i$ it is required that $P_j \subseteq \bigcup_{k=1}^{i} S_k$, which ensures that the precedence constraints between the tasks are respected.

2.2. Reversal of Problem Instances

Given a problem instance (T, S, W, G), the corresponding reverse problem instance (T, S, W, G^r) is obtained by inverting all the arcs in the precedence graph G. Each solution $\tau^r = (\pi^r, \mathcal{S}^r)$ to the reverse problem instance (T, S, W, G^r) can be converted into a solution $\tau = (\pi, \mathcal{S})$ to the original problem instance (T, S, W, G) as follows:

$$\pi_i := \pi^r_{m-i-1} \quad \text{for } i = 1, \ldots, m \tag{11}$$
$$S_i := S^r_{m-i-1} \quad \text{for } i = 1, \ldots, m \tag{12}$$

It is known from the literature (see, for example, [2]) that tackling the reverse problem instance may lead an exact algorithm faster to an optimal solution,

[1]Note that we refer to the variable cycle time of the IP model as z, while given fixed cycle times are denoted by C.

respectively, may provide a better heuristic solution when tackled with the same heuristic as the original problem instance.

3. The Algorithm

The basic component of our algorithm for the ALWABP is beam search (BS). Even though BS has proved to be a useful heuristic tool especially in the context of scheduling problems (see, for example, [10, 11, 12]), only very few applications to other types of problems exist (see, for example, [13, 14]). BS is an incomplete derivative of branch & bound that was introduced in [8]. In the following we briefly describe the working of one of the standard versions of BS. The central idea behind BS is to allow the extension of partial solutions in several possible ways. The algorithm keeps a set B of at most k_{bw} partial solutions. Hereby, B is the so-called beam, and k_{bw} is refered to as the beam width. At each step, the algorithm chooses at most k_{ext} feasible extensions of each partial solution in B. This choice of feasible extensions is done deterministically by means of a greedy function that assigns a weight to each feasible extension. At the end of each step, the algorithm creates a new beam B by selecting up to k_{bw} partial solutions from the set of chosen feasible extensions. For that purpose, BS algorithms calculate—in the case of minimization—a lower bound value for each chosen extension. Only the maximally k_{bw} best extensions—with respect to the lower bound—are chosen to constitute the new set B. Finally, the best found complete solution (if any) is returned. Crucial components of BS are the underlying constructive heuristic that defines the feasible extensions of partial solutions and the lower bound function for evaluating partial solutions.

In the following we first present a description of our implementation of BS for the ALWABP. Afterwards we describe the algorithmic scheme in which this beam search component is used for obtaining good solutions to the ALWABP.

3.1. Beam Search

The BS component described in this section—and pseudo-coded in Algorithm 1—is the main component of our algorithm for the ALWABP. As input the algorithm requires a problem instance (T, S, W, G), a fixed cycle time C, a beam width k_{bw}, and a maximal number of extensions k_{ext}. In short, BS tries to generate at least one feasible solution with m or less work stations, while respecting cycle time C. As mentioned before, the central idea behind BS is to allow the extension of partial solutions in several possible ways. At each step the algorithm extends each partial solution from beam B in a maximum number of ways. An extension is generated by, first, assigning an unassigned worker to the next empty work station and, second, by assigning a set of so-far unassigned tasks to this worker such that the given cycle time C is not surpassed and the precedence constraints between the tasks are respected (see lines 13-14 of Algorithm 1). More specifically, given a partial solution τ' with $l - 1 < m$ work stations already filled, the algorithm generates for each unassigned worker a maximum of k_{ext} extensions. In order to avoid having to enumerate all possible

Algorithm 1 Beam search

```
1: input: an instance $(T, S, W, G)$, a fixed cycle time $C$, a beam width $kbw$, and $k_{\text{ext}}$
2: $l := 0$
3: $\tau := (\pi, \mathcal{S})$ where $\pi$ and $\mathcal{S}$ are empty
4: $B := \{\tau\}$
5: $B_{\text{compl}} := \emptyset$
6: while $B \neq \emptyset$ do
7:    $B_{\text{ext}} := \emptyset$
8:    $l := l + 1$
9:    for all $\tau \in B$ do
10:      for all unassigned workers $w$ with respect to solution $\tau$ do
11:         for $i = 1, \ldots, k_{\text{ext}}$ do
12:            $\tau' := \tau$ {copy partial solution $\tau$ into $\tau'$}
13:            $\pi'_l := w$
14:            $S'_l :=$ ExtendPartialSolution($\tau', l, w, C$) {see Algorithm 2}
15:            if solution $\tau'$ is complete (that is, all tasks are assigned) then
16:               $B_{\text{compl}} := B_{\text{compl}} \cup \{\tau'\}$
17:            else
18:               if $l < m$ then
19:                  $B_{\text{ext}} := B_{\text{ext}} \cup \{\tau'\}$
20:               end if
21:            end if
22:         end for
23:      end for
24:   end for
25:   $B \leftarrow$SelectSolutions($B_{\text{ext}}$,$k_{\text{bw}}$)
26: end while
27: output: If $B_{\text{compl}} \neq \emptyset$ the output is TRUE, otherwise FALSE
```

extensions of a partial solution, our algorithm produces extensions in a (partially) probabilistic way rather than in the usual deterministic manner. Each generated extension (partial solution) is either stored in set B_{compl} in case it is a complete solution, or in set B_{ext} otherwise (see lines 15-21 of Algorithm 1). However, a partial solution is only stored in set B_{ext} in case less than m work stations have been filled already. At the end of each step, the beam search algorithm creates a new beam B by selecting up to k_{bw} (called the *beam width*) solutions from the set of further extensible solutions B_{ext} (see line 25 of Algorithm 1). This is done in function SelectSolutions(B_{ext},k_{bw}) by means of a lower bound LB($\cdot$). In the following we outline in more detail the extension of partial solutions and the working of function SelectSolutions(B_{ext},k_{bw}).

Algorithm 2 Function $\mathsf{ExtendPartialSolution}(\tau', l, w, C)$ of Algorithm 1

```
1: input: A partial solution $\tau'$, the index $l$ of the work station to be filled, the worker $w$ assigned to work station $l$, and the cycle time $C$
2: $S'_l := \emptyset$
3: $T' := \{i \in T \mid i \notin \bigcup_{j=1}^{l} S'_j, P_i \subseteq \bigcup_{j=1}^{l} S'_j, t_{iw} < \infty, t_{iw} + c_{\text{rem}} \leq C\}$
4: $c_{\text{rem}} := C$
5: while $T' \neq \emptyset$ do
6:     $j :=\mathsf{ChooseTask}(T', c_{\mathsf{rem}})$
7:     $c_{\text{rem}} := c_{\text{rem}} - t_{jw}$
8:     $T' := \{i \in T \mid i \notin \bigcup_{j=1}^{l} S'_j, P_i \subseteq \bigcup_{j=1}^{l} S'_j, t_{iw} < \infty, t_{iw} + c_{\text{rem}} \leq C\}$
9:     $S'_l := S'_l \cup \{j\}$
10: end while
11: output: Filled work station $S'_l$
```

3.1.1. Generating Extensions of Partial Solutions

The generation of an extension of a partial solution $\tau' = (\pi', \mathcal{S}')$ with $l-1$ work stations already filled takes place in lines 13-14 of Algorithm 1. In this context, let us denote by $\overline{W} \subseteq W$ the set of workers that are not assigned to any of the first $l-1$ work stations of partial solution τ'. The generation of an extension works as follows. First, a worker $w \in \overline{W}$ is assigned to work station l of τ', that is, $\pi'_l := w$. Second, unassigned tasks are iteratively assigned to worker π'_l—that is, added to S'_l—until the sum of their worker-dependent processing times is such that no other task can be added to S'_l without exceeding the given cycle time C. This procedure is pseudo-coded in Algorithm 2. At each step, T' denotes the set of so-far unassigned tasks that may be added to S'_l without violating any constraints. The definition of this set of *available tasks* is given in line 3, respectively 8, of Algorithm 2.

It remains to outline the implementation of function $\mathsf{ChooseTask}(T', c_{\mathsf{rem}})$ of Algorithm 2. For that purpose let us first define the following subset of T':

$$T^{\text{sat}} := \{i \in T' \mid t_{iw} + c_{\text{rem}} = C\} \tag{13}$$

Hereby, remember that w was the worker assigned to work station l. The definition in Eq. 13 is such that T^{sat} contains all tasks that *saturate*, in terms of processing time, work station l. Next, a sophisticated greedy function is employed for assigning a value $\eta_i > 0$ to all tasks $i \in T'$. Note that the definition of these greedy values follows below.

The first action for choosing a task from T' consists in flipping a coin for deciding if the choice is made deterministically, or probabilistically. In case of a deterministic extension, there are two possibilities. First, if $T^{\text{sat}} \neq \emptyset$, the best task from T^{sat} is chosen, that is, the task with maximal greedy value among all tasks in T^{sat}. Otherwise, we choose the task with maximal greedy value from T'. In case of a probabilistic decision, a task from T' is chosen using the following probability distribution:

$$\mathbf{p}(i) := \frac{\eta_i}{\sum\limits_{j \in T'} \eta_j} \quad, \forall i \in T' \tag{14}$$

For completing the description of the working of function $\mathsf{ChooseTask}(T', c_{\mathsf{rem}})$, it remains to outline the definition of the greedy values η_i, $\forall i \in T'$. The greedy value η_i of a task $i \in T'$ is composed of a static term α_i and a dynamic term β_i:

$$\eta_i := \alpha_i \cdot \beta_i \tag{15}$$

For each application of BS, the static term α_i for each $i \in T$ is pre-computed as follows:

$$\gamma_i := \kappa_1 \cdot \left(\frac{t_{wi}}{C}\right) + \kappa_2 \cdot \left(\frac{|\mathrm{Suc}_i^{\mathrm{all}}|}{\max_{1 \leq j \leq n} |\mathrm{Suc}_j^{\mathrm{all}}|}\right) \quad, \forall i \in T \tag{16}$$

Remeber again, that w is the worker assigned to work station l. In Eq. 16, $\mathrm{Suc}_i^{\mathrm{all}}$ denotes the set of all tasks that can be reached from i in the precedence graph G via a directed path. Moreover, κ_1 and κ_2 are parameters with values in $[-1, 1]$. Given the γ_i values, the static term α_i is derived as

$$\alpha_i := \frac{\gamma_i - \gamma_{\mathrm{min}} + 1}{\gamma_{\mathrm{max}}} \quad \forall\, i \in T \ , \tag{17}$$

where γ_{min}, respectively γ_{max}, denote the minimum, respectively maximum, values of all γ_i. Interestingly, for obtaining well-working α_i values, parameters κ_1 and κ_2 have to be chosen in a problem-instance-dependent way. However, in preliminar experiments we have not been able to derive parameter values that work well over the whole range of problem instances. Therefore, both parameter values are chosen randomly from $[-1, 1]$ for each calculation of the α_i values, that is, for each application of beam search.

In contrast to the static α_i values, values β_i are dynamic, that is, they depend on the current partial solution. Given a partial solution τ' with $l - 1$ work stations already filled, remember that $\overline{W} \subseteq W$ was defined as the set of workers that have not been assigned to any of the first $l - 1$ work stations of τ'. Then,

$$\begin{aligned} \beta_i^{\mathrm{min}} &:= \min\nolimits_{h \in \overline{W}} \{t_{ih} \mid t_{ih} < \infty\} \ , \forall i \in T' \quad (18) \\ \beta_i^{\mathrm{max}} &:= \max\nolimits_{h \in \overline{W}} \{t_{ih} \mid t_{ih} < \infty\} \ , \forall i \in T' \ . \quad (19) \end{aligned}$$

Given these definitions, and given that worker w is assigned to work station l of partial solution τ', terms β_i are defined as follows:

$$\beta_i := \frac{\beta_i^{\mathrm{max}} - t_{iw} + 1}{\beta_i^{\mathrm{max}} - \beta_i^{\mathrm{min}} + 1} \quad, \forall i \in T' \tag{20}$$

In other words, in contrast to the first term in the definition of γ_i (see Eq. 16) which takes into account the absolute processing time of a task, the term β_i measures the *goodness* of the processing time of a task i when executed by worker w in comparison to the processing times of task i when executed by any of the other so-far unassigned workers.

3.1.2. Lower Bound

The last action of BS at each step consists in determining the beam B of the next step. This new beam is chosen from B_{ext} in function $\mathsf{SelectSolutions}(B_{\text{ext}}, k_{\text{bw}})$ of Algorithm 2. First, the solutions in B_{ext} are ranked with respect to increasing lower bound values. Then, the $\min\{k_{\text{bw}}, |B_{\text{ext}}|\}$ highest ranked partial solutions from B_{ext} are chosen. For the purpose of ranking we used a relatively simple lower bound, in the following denoted by $\text{LB}(\cdot)$. Let us denote by $\overline{T} \subseteq T$ the set of tasks that have not yet been assigned to workers in partial solution τ'. Then:

$$\text{LB}(\tau') = \left\lceil \frac{\sum_{i \in \overline{T}} \min_{h \in \overline{W}} \{t_{ih}\}}{C} \right\rceil \tag{21}$$

Remember that further above we defined as $\overline{W}$ the set of workers that—with respect to a partial solution τ'—are not yet assigned to work stations.

3.2. Algorithmic Scheme

The algorithmic scheme which makes use of BS is related to the one proposed in [5]. The pseudo-code of this scheme, which we labelled iterated beam search (IBS), is given in Algorithm 3. The first step consists in determining a starting cycle time C. For that purpose we use the same lower bound for an optimal solution as given in [5]. More specifically, funcion $\mathsf{DetermineStartingCycleTime}()$ of Algorithm 3 determines the starting cycle time as $C = \max\{C1, C2\}$, where

$$C1 \quad := \quad \max_{i \in T}\{\min_{h \in W}\{t_{ih}\}\} \ , \tag{22}$$

$$C2 \quad := \quad \left\lceil \frac{\sum_{i \in T} \min_{h \in W}\{t_{ih}\}}{m} \right\rceil \ . \tag{23}$$

The algorithm works in two phases. In the first phase (see lines 3-12 of Algorithm 3) the algorithm tries to find very quickly a first cycle time C for which a valid solution can be found. For this purpose, BS is applied with the settings $k_{\text{bw}} = 1$ and $k_{\text{ext}} = 1$. This first phase ends for all instances considered in Section 4 after a fraction of a second. The second phase of the algorithm iteratively tries to find a valid solution for the next smaller cycle time. In this phase, the algorithm disposes over a certain time limit for each considered cycle time. Remember that BS works partially probabilistic. Therefore, it can be applied in a repeated way with potentially different outcomes. The first five percent of the mentioned time limit are spent with BS applications that use the settings $k_{\text{bw}} := 10$ and $k_{\text{ext}} := 1$. This is done with the intention of rapidly finding a feasible solution for the given cycle time, if possible. If BS in not able to solve the given cycle time with these settings, the remaining 95% of the time limit are

Algorithm 3 Iterated beam search (IBS) for the ALWABP

```
1: input: an instance $(T, S, W, G)$
2: $C :=$ DetermineStartingCycleTime()
3: $k_{\text{bw}} := 1$, $k_{\text{ext}} := 1$
4: $success :=$ FALSE
5: while not $success$ do
6:    $success :=$ BeamSearch($(T, S, W, G), C, k_{\text{bw}}, k_{\text{ext}}$) {original instance}
7:    if not $success$ then
8:       $success :=$ BeamSearch($(T, S, W, G^r), C, k_{\text{bw}}, k_{\text{ext}}$) {reverse instance}
9:       if not $success$ then $C := C + 1$ end if
10:   end if
11: end while
12: $C := C - 1$
13: $stop :=$ FALSE
14: while not $stop$ do
15:    $success :=$ FALSE
16:    while time limit not reached and not $success$ do
17:       if within 5% of time limit then $k_{\text{bw}} := 10$, $k_{\text{ext}} := 1$ else $k_{\text{bw}} := 100$,
          $k_{\text{ext}} := 10$ end if
18:       $success :=$ BeamSearch($(T, S, W, G), C, k_{\text{bw}}, k_{\text{ext}}$) {original instance}
19:       if not $success$ then
20:          $success :=$ BeamSearch($(T, S, W, G^r), C, k_{\text{bw}}, k_{\text{ext}}$) {reverse instance}
21:       end if
22:    end while
23:    if $success$ then $C := C - 1$ else $stop :=$ TRUE end if
24: end while
25: $C := C + 1$
26: output: cycle time $C$
```

spent with BS applications using the setting $k_{\text{bw}} := 100$ and $k_{\text{ext}} := 10$. With these settings BS is much slower. However, the probability of finding feasible solutions is much higher than with the settings described before. The second phase of the algorithm ends when the time limit has passed without having found a feasible solution for the considered cycle time.

4. Computational results

We implemented IBS in ANSI C++ using GCC 3.4.0 for compiling the software. Our experimental results were obtained on a PC with an AMD64X2 4400 processor and 4 Gb of memory. In the following we first describe the set of benchmark instances that we used for the experimental evaluation. Afterwards we will focus on the results of the proposed algorithm.

4.1. Benchmark Instances

For the experimental evaluation of IBS we used a set of 320 benchmark instances proposed by Chaves et al. in [7]. This benchmark set was constructed following a two-level five factors full factorial scheme from the well-known classical SALBP collection of Hoffmann [15]. The original problem instances for generating the benchmark set were selected from that collection, so that problems with low and high *order strength* (OS), which measures the structural properties of the precedence graph, and problems with low and high number of tasks were included. More specifically, the selected problem instances selected from [15] were Roszieg, Heskia, Tonge, and Wee-mag. Table 1 shows the characteristics of these problems.

Table 1: The characteristics of the selected problem instances.

Instance	**Number of tasks**	**Order strength**
Roszieg	25 tasks (low)	71.67 (high)
Heskia	28 tasks (low)	22.49 (low)
Tonge	70 tasks (high)	59.42 (high)
Wee-Mag	75 tasks (high)	22.67 (low)

The generation of ALWABP instances from these four problem instances was done as follows. Only the precedence graph was preserved from each original problem instances. The original processing times of the tasks were used as processing times concerning the first worker. The processing times concerning the remaining workers were randomly generated on the basis of the original processing times. From the available experience on sheltered work centers for the Disabled, the upper bound for these randomly generated processing times should not be greater than three times the original processing times. In case the processing time of a worker for a certain task is greater than this upper bound, it is assumed that this task should not be assigned to him or her. The worker is then assigned an infinite time for that task (which means that the respective task is incompatible with this worker). The problem instances were generated according to the following three factors (in addition to the two factors *number of tasks* and *order strength*):

- The high, respectively low, relation between the number of tasks and the number of workers (size of the task-worker matrix).
- The high, respectively low, variability of processing times for the different workers.
- The high, respectively low, percentage of task-worker incompatibilities defined a priori.

Concerning the first one of these three factors, *high* refers to a number of tasks **four** times higher than the number of workers, while *low* refers to a number of tasks **seven** times higher than the number of workers. The different processing times for each task were randomly generated from a uniform distribution with a range selected according to the original processing time (as described above). Given a task i with its original processing time t_i, *high* refers to the range $[1, 3 \cdot t_i]$ and *low* refers to the range $[1, t_i]$. And finally, the *high*, respectively *low*, percentage of task-worker incompatibilities was set to 20%, respectively 10%. Hence, globally the benchmark was created according to the following five factors: (1) number of tasks, (2) order strength, (3) mean number of tasks per worker, (4) variability of task times, and (5) percentage of task-worker incompatibilities. As high and low levels are defined for each factor we count 32 combinations. Generating 10 problems for each combination resulted in a benchmark of 320 ALWABP instances of varying characteristics. Note that for each of the four original SALBP instances, 80 ALWABP instances were obtained.

4.2. Experimental Results

We applied IBS for 20 times to each of the 320 problem instances. As a time limit for each considered cycle time (see Algorithm 3) we used 120 seconds. In the following we present our results in the same way as done in [6]. Moreover, we compare our results to the results of the current state-of-the-art algorithm as published in [6]. This algorithm is labelled CS (for "clustering search"). The results are shown in Tables 2 to 5. Each of these tables presents the results for the 80 problem instances obtained from one of the four original SALBP instances (Roszieg, Heskia, Tonge, and Wee-mag). Moreover, the results are presented in a summarized way, as averages over the 10 instances for the same combination of factors (3), (4) and (5). This results in eight groups of 10 instances for each of the four original SALBP instances. Groups 1-4 are characterized by a low number of workers, whereas groups 5-8 contain instances with a high number of workers. Furthermore, groups 1, 2, 5, and 6 are characterized by a low variability of processing times, whereas groups 3, 4, 7, and 8 contain instances with a high variability of processing times. Finally, groups 1, 3, 5, and 7 are characterized by a low percentage of worker incompatibilities, while the remaining groups consist of instances with a high percentage of worker incompatibilities.

The results of the two algorithms (IBS and CS) are presented in three columns. The first of these column—with heading **best**—shows the average over the best solutions found for the 10 instances of the respective groups within 20 applications of each algorithm to each instance. This measure is henceforth refered to as "best-performance". The second column with heading **avrg** provides the "average-perfornamce", that is, the average over the averages of each algorithm for the 10 instances of each group. Finally, the last column with heading **time** gives the average times when the best solutions of all the runs were found (again averaged over the 10 instances per group). The last row of each table provides averages over the 8 groups.

The results show, first, that in terms of average performance IBS is—with the exception of one case—always better than CS. The exception is Roszieg, group 4, where both algorithms show the same average-performance. Especially when the number of tasks grows (see Tonge and Wee-mag families) the average performance of IBS is considerably better than the one of CS. Notice that in the cases of Tonge and Wee-mag, the average-performance of our algorithm is even better than the best-performance of CS. The best-performances of the two algorithms are equal when instances with a low number of tasks are concerned (Roszieg and Heskia). However, in the cases of Tonge and Wee-mag, IBS obtains in all 16 cases (two times eight groups) a new best-performance (as well as a new best average-performance). This is indicated by the asterisk in the columns with the heading **best-known**. The asterisk indicates that this performance was obtained for the first time in the literature. Interestingly we were not able to detect any decrease in performance for any of the 8 different groups. This means that—in comparison to CS—IBS works well regardless of the instance characteristics, even though we can notice that the absolute improvement of IBS over CS is higher for instances with a high number of workers (groups 5-8). Concerning robustness we can say that IBS is much more robust than CS. This is indicated by the average-performances. In the case of CS the difference between the best-performances and the average-performances is much higher than in the case of IBS. Therefore, we may say that IBS is more robust than CS.

Concerning the processing times, CS was evaluated on computer with a 2.6 GHz Pentium 4 processor and 1 Gb of memory. Being conservative, we might assume that the processor that was used to evaluate IBS is double as fast as the processor used to evaluate CS. In this case, beam search was faster than CS on the 80 instances derived from Roszieg. However, beam search was (up to three times) slower than CS on the remaining instances. However, as the ALWABP is not a time-critical problem, the focus is on obtaining good solution rather than on developing the fastest algorithm.

Finally, let us mention that the results of the small instances (Roszieg and Heskia) have been proved to be optimal by solving the IP formulation presented in Section 2 with CPLEX. In contrast, the results of CPLEX for the large problem instances (Tonge and Wee-mag families) are characterized by very high gaps between upper and lower bounds even after very long running times. The values of the best solutions for each of the 320 instances (optimal in case of Roszieg and Heskia, and best-known in case of Tonge and Wee-mag) are given in the tables in Appendix A.

5. Conclusions and Outlook to the Future

In this work we presented an algorithm based on beam search for the assembly line worker assignment and balancing problem (ALWABP). The results have shown that our algorithm is currently a state-of-the-art algorithm for this problem. In the future we plan to apply similar techniques to other assembly line balancing problems with additional constraints.

Table 2: Results for the Roszieg family.

Group	**best-known**	**IBS**			**CS**		
		best	avrg	time	best	avrg	time
1	20.1	20.1	20.1	0.01	20.1	20.2	0.8
2	31.5	31.5	31.5	0.09	31.5	32.5	0.9
3	28.1	28.1	28.1	0.13	28.1	28.5	0.6
4	28.0	28.0	28.0	0.00	28.0	28.0	0.2
5	9.7	9.7	9.7	0.01	9.7	10.7	1.3
6	11.0	11.0	11.0	0.02	11.0	12.1	1.4
7	16.0	16.0	16.0	0.01	16.0	16.9	1.5
8	15.1	15.1	15.1	0.01	15.1	15.6	1.9
average	**19.94**	**19.94**	**19.94**	**0.04**	**19.94**	**20.57**	**1.09**

Table 3: Results for the Heskia family.

Group	**best-known**	**IBS**			**CS**		
		best	avrg	time	best	avrg	time
1	102.3	102.3	102.3	8.16	102.3	102.8	1.3
2	122.6	122.6	122.6	2.98	122.6	123.8	1.4
3	172.5	172.5	172.5	5.63	172.5	175.5	1.7
4	171.2	171.2	171.27	5.21	171.2	171.7	1.4
5	34.9	34.9	34.9	1.09	34.9	37.8	4.4
6	42.6	42.6	42.6	2.54	42.6	44.7	3.4
7	75.2	75.2	75.2	1.67	75.2	77.7	2.9
8	67.2	67.2	67.2	2.51	67.2	70.7	3.6
average	**98.56**	**98.56**	**98.57**	**3.72**	**98.56**	**100.59**	**2.51**

Table 4: Results for the Tonge family.

Group	**best-known**	**IBS**			**CS**		
		best	avrg	time	best	avrg	time
1	$*94.9$	94.9	96.66	86.38	96.7	116.6	64.0
2	$*110.2$	110.2	111.53	92.17	116.0	141.8	64.6
3	$*165.0$	165.0	168.02	150.28	167.7	199.4	66.2
4	$*170.0$	170.0	171.41	149.50	174.0	206.0	65.7
5	$*33.1$	33.1	34.17	87.98	41.3	51.3	101.1
6	$*40.0$	40.0	40.96	70.50	48.5	61.6	105.3
7	$*66.4$	66.4	67.89	124.28	77.8	93.0	100.1
8	$*64.7$	64.7	66.59	156.42	77.9	95.6	100.3
average	**$*$93.04**	**93.04**	**94.65**	**114.69**	**99.99**	**120.64**	**83.42**

References

[1] S. Gosh, R. J. Gagnon, A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems, International Journal of Production Research 27 (1989) 637–670.

[2] A. Scholl, C. Becker, State-of-the-art exact and heuristic solution proce-

Table 5: Results for the Wee-mag family.

Group	best-known	IBS			CS		
		best	avrg	time	best	avrg	time
1	∗28.7	28.7	29.71	104.91	29.0	32.7	94.3
2	∗33.6	33.6	34.91	84.93	34.6	38.4	91.4
3	∗50.1	50.1	51.6	160.33	50.8	56.7	96.0
4	∗48.6	48.6	50.44	143.34	49.6	55.6	103.9
5	∗10.3	10.3	10.67	57.05	13.1	20.9	141.2
6	∗11.9	11.9	12.35	60.24	14.6	18.2	155.2
7	∗18.2	18.2	18.96	71.36	21.2	27.1	148.0
8	∗18.1	18.1	18.85	90.00	21.6	26.8	140.6
average	**∗27.44**	**27.44**	**28.44**	**96.52**	**29.31**	**34.56**	**121.31**

dures for simple assembly line balancing, European Journal of Operational Research 168 (3) (2006) 666–693.

[3] C. Miralles, J. P. Garcia-Sabater, C. Andres, M. Cardos, Advantages of assembly lines in sheltered work centres for disabled, International Journal of Production Economics 110 (2007) 187–197.

[4] P. Kotler, N. Lee, Corporate Social Responsibility: Doing the Most Good for Your Company and Your Cause, Wiley & Sons, Hoboken, NJ, 2005.

[5] C. Miralles, J. P. Garcia-Sabater, C. Andres, M. Cardos, Branch and bound procedures for solving the assembly line worker assignment and balancing problem. application to sheltered work centres for disabled, Discrete Applied Mathematics 156 (2008) 352–367.

[6] A. A. Chaves, L. A. Nogueira Lorena, C. Miralles, Hybrid metaheuristic for the assembly line worker assignment and balancing problem, in: M. J. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, A. Schaerf (Eds.), Proceedings of HM 2009 – Sixth International Workshop on Hybrid Metaheuristics, Vol. 5818 of Lecture Notes in Computer Science, Springer Verlag, Berlin, Germany, 2009, pp. 1–14.

[7] A. A. Chaves, C. Miralles, L. A. Nogueira Lorena, Clustering search approach for the assembly line worker assignment and balancing problem, in: M. H. Elwany, A. B. Eltawil (Eds.), Proceedings of ICC&IE 2007 – 37th International Conference on Computers and Industrial Engineering, 2007, pp. 1469–1478.

[8] P. S. Ow, T. E. Morton, Filtered beam search in scheduling, International Journal of Production Research 26 (1988) 297–307.

[9] C. Blum, Beam-ACO for simple assembly line balancing, INFORMS Journal on Computing 20 (4) (2008) 618–627.

[10] I. Sabuncuoglu, M. Bayiz, Job shop scheduling with beam search, European Journal of Operational Research 118 (1999) 390–412.

[11] M. Ghirardi, C. N. Potts, Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach, European Journal of Operational Research 165 (2) (2005) 457–467.

[12] J. M. S. Valente, R. A. F. S. Alves, Filtered and recovering beam search algorithms for the early/tardy scheduling problem with no idle time, Computers & Industrial Engineering 48 (2) (2005) 363–375.

[13] G.-C. Lee, D. L. Woodruff, Beam search for peak alignment of NMR signals, Analytica Chimica Acta 513 (2) (2004) 413–416.

[14] C. Blum, M. J. Blesa, M. López Ibáñez, Beam search for the longest common subsequence problem, Computers & Operations Research 36 (12) (2009) 3178–3186.

[15] T. R. Hoffmann, Assembly line balancing: A set of challenging problems, International Journal of Production Research 28 (1990) 1807–1815.

Appendix A

Table 6: Values of optimal solutions for the Roszieg family.

Group	**Instance number**									
	1	2	3	4	5	6	7	8	9	10
1	20	22	18	18	17	24	21	20	22	19
2	30	27	76	25	26	22	22	20	27	40
3	28	30	26	33	28	27	21	28	27	33
4	31	29	32	27	27	29	27	28	21	29
5	10	10	10	9	12	9	10	8	10	9
6	11	10	10	10	11	13	13	11	12	9
7	16	13	19	16	14	17	17	16	15	17
8	15	16	16	16	16	17	13	14	14	14

Table 7: Values of optimal solutions for the Heskia family.

Group	**Instance number**									
	1	2	3	4	5	6	7	8	9	10
1	94	95	102	103	92	98	116	86	95	142
2	169	107	108	96	130	117	146	132	101	120
3	200	147	186	181	142	194	149	191	170	165
4	204	147	211	127	181	179	191	152	167	153
5	35	40	35	30	40	29	25	43	38	34
6	51	50	52	33	38	34	42	39	59	28
7	66	56	69	126	107	39	87	62	61	79
8	91	65	73	57	65	70	71	58	46	76

Table 8: Values of the best-known solutions for the Tonge family (as obtained by IBS).

Group	**Instance number**									
	1	2	3	4	5	6	7	8	9	10
1	93	94	106	112	92	90	95	108	73	86
2	112	116	105	93	95	101	123	113	128	116
3	171	161	148	150	174	173	149	182	189	153
4	175	180	157	181	152	169	167	153	183	183
5	29	33	33	34	31	38	32	35	34	32
6	38	43	36	40	40	40	42	42	37	42
7	63	68	69	97	65	69	52	63	55	63
8	57	61	72	68	69	59	62	67	66	66

Table 9: Values of the best-known solutions for the Wee-mag family (as obtained by IBS).

Group	**Instance number**									
	1	2	3	4	5	6	7	8	9	10
1	28	28	27	33	32	26	27	31	27	28
2	32	33	32	33	34	32	34	37	33	36
3	50	48	48	52	49	47	56	51	45	55
4	51	49	52	58	47	49	42	47	50	41
5	10	10	10	12	11	10	11	10	9	10
6	12	10	11	12	12	13	12	12	14	11
7	16	18	19	17	18	18	20	15	20	21
8	19	16	17	21	19	16	18	18	20	17