A fast and effective heuristic for smoothing workloads on assembly lines: algorithm design and experimental analysis

Öncü Hazir, Maher Agi, Jeremy Guérin

A Fast and Effective Heuristic for Smoothing Workloads on Assembly Lines: Algorithm Design and Experimental Analysis

Öncü Hazır[a,*], Maher Agi[a], Jérémy Guérin[a]

[a]*Rennes School of Business, 2 Rue Robert d'Arbrissel, Rennes, France*

Abstract

Workload smoothing on assembly lines, which aims to evenly assign tasks to stations, supports workforce planning and resource optimization. In this paper, we study smoothing assembly lines and develop a problem-specific heuristic to efficiently solve large-sized instances. To build solutions, the algorithm uses a number of well-known priority rules for task assignment in conjunction with a probabilistic decision-making procedure for closing workstations. We next conduct an experimental design to select the best performing priority rules and tuning of the probabilistic decision-making procedure. The efficiency of our algorithm is tested and demonstrated through an extensive experimental study.

Keywords: Assembly line balancing, Combinatorial optimization, Heuristics, Priority rules, Workload Smoothing

*Corresponding author.
Email addresses: oncu.hazir@rennes-sb.com (Öncü Hazır), maher.agi@rennes-sb.com (Maher Agi), guerinj17@gmail.com (Jérémy Guérin)

1. Introduction

Assembly lines (AL) have been traditionally used since decades in many industries such as automotive and electronics, as they enable mass production of standardized products efficiently. These systems are composed of serially located workstations where workers continuously perform operations. Designing and balancing different line configurations and optimizing their capacity usage have been taking the attention of the researchers and a vast literature has already developed in this field. The related studies can be classified according to the number of product models assembled: simple assembly line balancing problem (SALBP), mixed assembly line balancing problem (MALBP), and multi-model assembly line balancing problem (MMALBP) lines. Simple lines produce one homogeneous product and require similar production processes for all the production. In mixed-model lines, several versions of the same product are assembled. In multi-model assembly lines, production processes might differ significantly and require additional set-ups (Scholl, 1999). In addition to the above classification, assembly line balancing problems can be classified according to the problem objective (Boysen et al., 2008; Battaïa and Dolgui, 2013). Minimizing the number of stations or the cycle time (respectively, type I and type II problems) are the most common objectives in the existing literature. In this research, we focus on simple lines and deal with the workload smoothing problem, which is referred to as the type III problem (Uğurdağ et al., 1997; Eswaramoorthi et al., 2012).

Workload smoothing, that consists of evenly assigning tasks to stations (Moodie and Young, 1965), has several benefits. It makes workforce and payroll planning easier and improves resource utilization. It also helps reduce

ergonomic risks and minimizes worker fatigue and accidents that could be caused by disproportionate workloads (Otto and Scholl, 2011; Groover, 2013; Finco et al., 2019). We refer to Groover (2013) for a discussion about the use of smoothing to improve general workstation design. Moreover, workload smoothing enhances work equity by dividing the work fairly among the workers (Rachamadugu and Talbot, 1991), and it has been found that in mixed-model assembly lines, smoothing reduces short-term work overloads (Emde et al., 2010). Despite all these important benefits in practice, research on workload smoothing, the type III problem, is scant compared to other problem types.

In the literature, both exact and heuristic approaches have been implemented for solving smoothing problems by using different objective functions. Pinnoi and Wilhelm (1997) use a branch & cut method to minimize the maximum idle time for a specified number of stations, and Azizoğlu and İmat (2018) employ a branch & bound algorithm to minimize the sum of squared workstations loads. Walter (2020) corrects the lower bound of Azizoğlu and İmat (2018) and proposes a tighter formulation. These exact method-based works emphasize the high complexity of the problem and the difficulty of solving it, as only small-sized instances which contain up to 30 tasks are solved to optimality.

Larger-sized instances (up to 148 tasks) are solved using heuristic methods, which are mainly metaheuristics. Kim et al. (1996) use genetic algorithms to minimize the mean squared deviation from the mean workstation load. Further, using the same method, Kim et al. (1998) minimize the mean absolute deviation (MAD), which measures the station loads' deviations from

the mean workload. Uğurdağ et al. (1997) focus on solving the type II problem; however, the second stage of their algorithm minimizes workload variance as a secondary objective. Nearchou (2011) addresses a bi-criteria problem with minimization of the cycle time and MAD using a particle swarm optimization-based method. Eswaramoorthi et al. (2012) design a two-stage heuristic procedure for minimizing a flow index measured by the root of the mean squared deviation from the takt time. Mozdgir et al. (2013) use Taguchi method for optimizing the smoothness index (SI) introduced by Moodie and Young (1965) and measured by the sum of squared deviations of workstation loads from the cycle time. Finally, Finco et al. (2019) use a mixed integer linear programming-based (MILP-based) method and a heuristic approach to minimize the SI's square root.

In this study, we develop a problem-specific heuristic procedure that solves large-sized instances (up to 1,000 tasks) efficiently. Our algorithm's efficiency is mainly due to the use of a new probabilistic rule for closing stations, that is specifically designed to achieve the smoothing objective. It is also enhanced by a thorough experimental design we conduct for selecting the best-performing combinations of priority rules used for task assignment and parameter values of the probabilistic rule used for closing stations.

Developing efficient heuristics contributes both to the practice and theory. Relatively efficient heuristics support practitioners in quickly generating reliable solutions and evaluating several alternatives. In addition, they can enhance the efficiency of several optimization methods, such as the branch and bound algorithms (Otto and Otto, 2014). The efficiency of our heuristic is demonstrated through extensive computational experiments that we real-

ize using well-known benchmark data sets from Scholl (1999) and Otto et al. (2013). At this point, we refer to the experimental analysis and comparison studies of Amen (2000), Pape (2015), Li et al. (2017a) and Li et al. (2017b) on heuristic algorithms for solving other line balancing problems.

The remainder of this paper is organized as follows. First, we present a formal definition and a mathematical model of the problem in section 2. Then, we build the solution algorithm and explain its components in section 3. Later, in section 4, we present the experimental analysis and computational results. Finally, conclusions and directions for future research are given in Section 5.

2. Problem Definition

We consider a simple assembly line in which a fixed number of tasks, (n), are undertaken on a given number of workstations, (K). Tasks are assigned to the stations so that a product could be assembled within a given cycle time, (C). In addition, precedence constraints, which define the processing order of the tasks, should be respected. A graph, $G = (N, A)$ where N is the set of nodes and $A \subseteq N \times N$ is the set of arcs represents these temporal relations. The objective is to balance the workloads on the K stations, formally, to minimize the sum of squared differences between stations' times and the cycle time (C). Any task (i) can be characterized by t_i, P_i, and F_i, which are, respectively, the task time, the set of its direct predecessors, and the set of its direct followers. Let w_k be the load of station k and x_{ik}, a decision variable that takes the value 1 when operation i is assigned to station k, the mathematical formulation of our problem is given as follows:

$$\text{Min} \sum_{k=1}^{K} (C - w_k)^2 \tag{1}$$

subject to

$$\sum_{k=1}^{K} x_{ik} = 1, \quad \text{for } i = 1, \ldots, n \tag{2}$$

$$\sum_{k=1}^{K} k x_{ik} \leq \sum_{k=1}^{K} k x_{jk}, \quad \forall (i,j) \in A \tag{3}$$

$$w_k = \sum_{i=1}^{n} t_i(x_{ik}) \leq C, \quad \text{for } k = 1, \ldots, K \tag{4}$$

$$x_{ik} \in \{0,1\} \qquad \forall i,k. \tag{5}$$

(1) represents the objective function. Equations (2) ensure that a unique station is assigned to each operation. Inequalities (3) and (4) represent, respectively, the precedence and cycle time constraints. This problem has shown to be strongly NP hard (Azizoğlu and İmat, 2018).

Note that the objective function in (1) corresponds to what is usually called the workload smoothness index (Scholl and Becker, 2006; Mozdgir et al., 2013). Azizoğlu and İmat (2018) show that minimizing this objective function is equivalent to minimizing the sum of the squares of the total workstations loads:

$$Z = \sum_{k=1}^{K} w_k^2 \tag{6}$$

Hereafter, we address minimizing this function. To solve this problem, we develop a heuristic.

3. Solution Approach and the Algorithm

In this section, we first introduce the solution approach that incorporates specific characteristics of the smoothing problem. Then, we present the algorithm that we have built based on the solution approach.

3.1. Solution Approach

To solve the problem, we build a multi-pass heuristic algorithm that uses several priority rules for assigning tasks to workstations combined with a probabilistic rule for closing the station. Using different priority rules helps generate alternative solutions. In addition, by employing a probabilistic rule, a different set of solutions is generated each time the algorithm is run.

3.1.1. Priority rules

We use a priority rule for choosing a task among several eligible tasks that all respect the precedence relationships. For this purpose, 21 priority rules listed in Otto and Otto (2014)[1] are tested, out of which 8 rules are definitely integrated in the algorithm based on the experimental design performed in section 4.1.

3.1.2. Probabilistic rule for closing workstations

As the overall objective of our work is to smooth the stations' workloads, it is obvious that it might not be optimal to fully use the station's capacity.

[1]See Appendix A for the list of all priority rules used here.

Indeed, even in cases where a workstation has enough idle time that could be used for assigning one or more remaining tasks, it might not be optimal to assign any of them to this workstation. Instead, achieving the smoothing objective might require closing the workstation and immediately opening the next one. Considering this problem-specific characteristic, we develop a probabilistic decision-making rule for closing workstations that aims to evenly distribute the workload among stations. This rule is based on a threshold value (T) of the workload specific to each workstation, that is equal to the average load on the currently open workstation and those which are still to be opened for task assignment. Formally, assuming that r is the currently open workstation, then, a load of $\sum_{i=1}^{n} t_i - \sum_{k=1}^{r-1} w_k$ time units is still to be assigned to this workstation and the remaining $K - r$ ones. Therefore, the threshold of station r that we denote by T_r is expressed as follows:

$$T_r = \frac{\left(\sum_{i=1}^{n} t_i - \sum_{k=1}^{r-1} w_k\right)}{K - r + 1} \tag{7}$$

Observe that T_r represents the ideal load of the current workstation r with regards to the smoothing objective. However, given the indivisibility of tasks, there is no guarantee that it will be possible to reach T_r on workstation r. Therefore, it is useful to have a rule that allows for closing the station once its workload reaches a value around T_r. For this purpose, we define a minimum closing load (MCL) for the workstation r based on the threshold value of this station T_r in conjunction with the probability function for closing the workstation as follows:

$$MCL_r = \alpha T_r \tag{8}$$

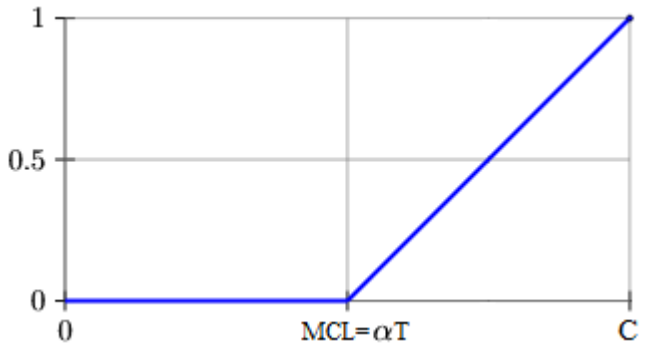

Figure 1: Probability of closing a station as a function of its workload in $[0, C]$.

$$\mathbb{P}(\text{close } r) = \begin{cases} 0 & \text{if } 0 \leq w_r \leq MCL_r \\ \dfrac{w_r - MCL_r}{C - MCL_r} & \text{if } MCL_r < w_r \leq C \end{cases} \tag{9}$$

Figure 1 shows the behavior of this probability function. Observe that our rule respects the workstation's capacity constraint as it imposes the decision of closing the workstation once the cycle time is reached. On the other hand, this rule prevents from closing the workstation before the MCL is reached.

We also note that the value of the coefficient α has a key role for determining when to close the workstation. Small values of α allow for closing the workstation at relatively low loads and result in a slow increase of the probability of closing the workstation following the assignment of tasks. On the other hand, higher values of α prevent from closing the workstation early and result in a quick increase of the probability of closing the workstation following the assignment of tasks. Hence, small values of α might lead to underutilization of stations capacity, so that the algorithm would end with

all stations being closed while there are still unassigned tasks. To deal with this issue, we integrate a repair procedure to the algorithm.

Alternatively, with higher values of α, there is a risk of poor quality solutions characterized by first workstations being overloaded while subsequent ones are underloaded. Considering these aspects, we make a comprehensive experimental design to choose the best performing values of α that allow for balancing the risks of underloading and overloading workstations. The implementation of this experimental design is detailed in section 4.1.

3.1.3. Repair Procedure

It is obvious that the solution approach developed above could result in infeasible solutions. Specifically, partial solutions could occur in which all workstations are closed while some tasks remain unassigned. In this case, a repair procedure is applied. This repair procedure uses a constructive algorithm and works as follows: the remaining unassigned tasks are ordered considering the precedence constraints and the current priority rule. Following this order, they are taken one by one and assigned to the earliest workstation that has enough idle time. If all the tasks can be assigned, then the partial solution is repaired. Otherwise, if the tasks cannot be assigned, then the partial solution is eliminated.

3.2. The Algorithm

The algorithm generates several sets of solutions by using different combinations of priority rules and α values used in the probability function. Then, the best solution is selected. Figure 2 shows how a solution is generated by the algorithm. It works as follows: each time a station is opened, the thresh-

old (T) and MCL values for this station are computed using equations (7) and (8). Then, tasks are ordered and assigned to stations using the priority rule while respecting the model constraints (precedence and cycle time). Each time a task is assigned to a station, the probability of closing the station is computed using (9), and the test of closing the station is performed as explained in section 3.1.2. An open station is closed if the result of the test for closing is positive or it is not possible to assign any of the remaining tasks while respecting the model constraints. Each time a station is closed, the value of the objective function is updated, and a new station is opened. The algorithm stops under one of the three following conditions:

- All stations are used and all tasks are assigned. In this case, the solution is considered feasible. This corresponds to the end instruction (1) in Figure 2.

- The last station (station no. K) is closed before the last task is assigned. Then, the solution is considered infeasible, and a repair procedure is applied (section 3.1.3). Depending on the repair procedure's success, we obtain the end instruction (1) or (3) in Figure 2.

- All tasks are assigned while there is still one or more empty stations. Then, the solution is considered as suboptimal (Azizoğlu and İmat, 2018) and is eliminated. This corresponds to the end instruction (2) in Figure 2.

For a better understanding of the problem and the solution approach used in our algorithm, we refer readers to the numerical example detailed in Appendix B.

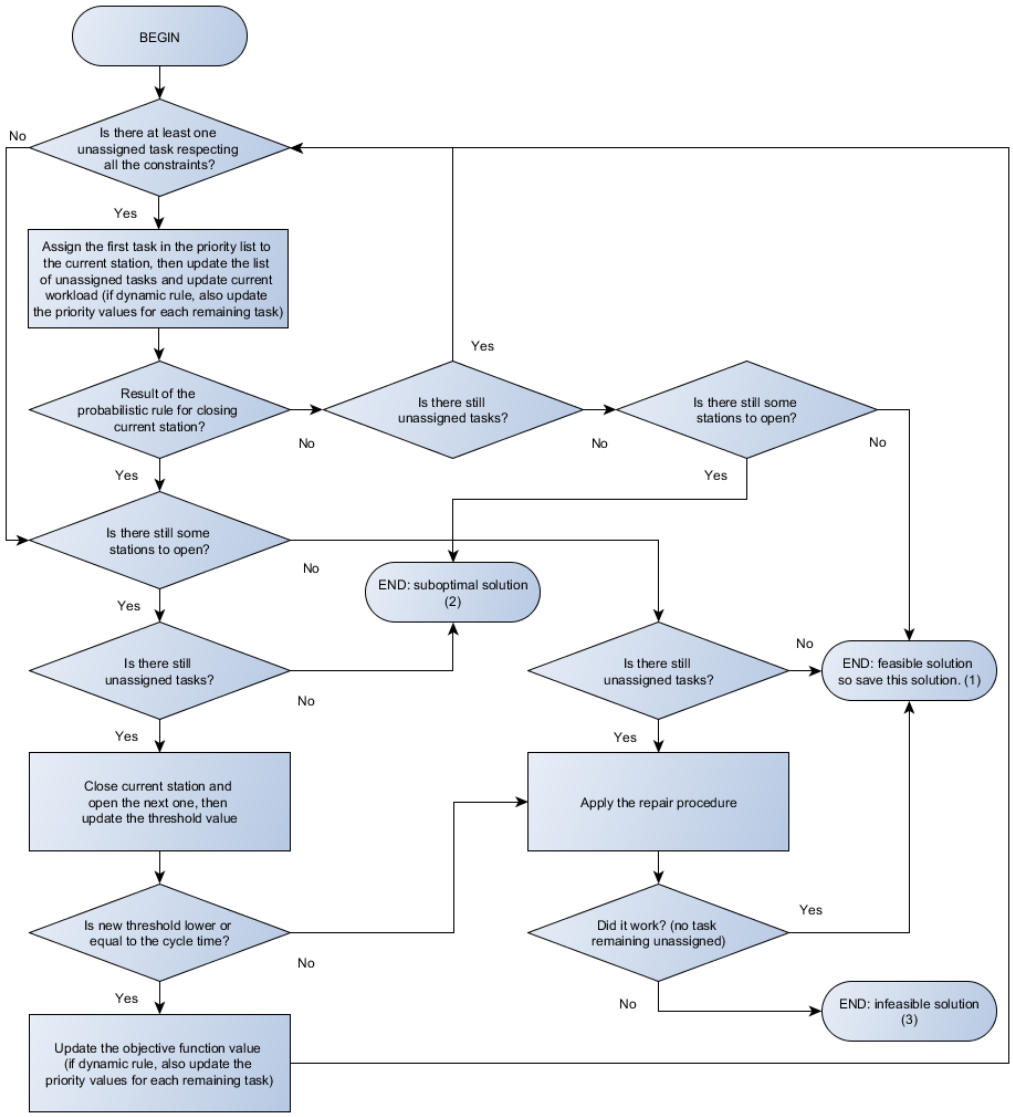

Figure 2: Flowchart of our heuristic procedure.

4. Experimental Analysis

In this section, we perform an experimental design for defining the heuristic's parameters values. Then we conduct extensive experiments to demonstrate the efficiency of our heuristic.

4.1. Parameter selection

To select the best-performing combination of priority rule $\times$ value of α, we conduct pilot tests using 39 small- to medium-sized instances from https://assembly-line-balancing.de. Optimal values of the objective function for these instances, which contain between 30 to 53 tasks each, are known. We perform experiments using the 21 priority rules in Appendix A along with the following values of α: 0; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; and 0.9. This results in 210 different combinations of priority rules and α values. Using each of these combinations, 1,000 solutions are generated by the algorithm for each one of the instances mentioned above. The best solution among the 1,000 is then selected, and the corresponding deviation from the optimal objective function value is computed. Appendix C shows the average deviation value out of the 39 instances for each combination. Using these results, we select the best-performing combinations: those that present an average deviation lower than 1.27%.

This yields 72 combinations to be definitely integrated in our algorithm. Examining these 72 combinations, we observe that they include nine values of α: 0; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; and 0.8 combined with the following eight priority rules: (1) total number of followers ($|F^*|$); (2) positional weight (PW^*); (3) latest station divided by number of followers (LdF); (4) recursive

cumulated positional weight (CPW); (5) critical path (CP); (6) longest path (LP); (7) recursive cumulated edges (RE); and (8) cumulated number of followers (CF).

Note that these priority rules are part of well-known priority rules used to solve SALBP-1 and SALBP-2 problems. Previous research has shown that the performance of the priority rules might significantly depend on the structural characteristics of the problem instances (Otto and Otto, 2014). For our problem, we emphasize the interaction of the rules with the probability function characterized by the parameter α. Therefore, the results are presented for pairs (Rule $\times$ α) in Appendix C. From Appendix C, we observe that the rules that use network structure characteristics (CP, CF, PW^*) perform well in the pilot tests, independently of the value of α. Moreover, the best results are usually achieved with $\alpha = 0.4, 0.5, 0.6, 0.7$. Therefore, the selected parameters reflect a balance of underloading and overloading and make use of the precedence network information.

4.2. Computational results

Otto et al. (2013) suggest the use of instances with 100 tasks for testing heuristic algorithms and instances with 1,000 tasks for evaluating the capacity of algorithms to solve real-size problems. Following this recommendation, we conduct computational experiments using three sets of instances selected from https://assembly-line-balancing.de/.

- We have 42 instances from Scholl (1999). This set comprises instances with 14 different values of n (from 21 to 297 tasks) and three different cycle times C for each value of n (the minimum, the median, and the

maximum of the values reported in the source files). We solve each of the 42 instances using the following three values of the number of stations: the optimal value of the SALBP-1 problem K, $K + 5\%$, and $K + 10\%$.

- Two sets of instances were selected randomly from Otto et al. (2013). This comprises:
  - Twenty-five instances with $n = 100$ tasks; and
  - twenty-five instances with $n = 1000$ tasks.

This results in a total of 276 instances used for testing our algorithm. For each instance of the first and second sets, 72,000 solutions are generated (1,000 solutions using each of the 72 combinations of rules $\times$ values of α), among which the algorithm returns the best solution. As for instances of the third set, a larger number of solutions (720,000) are generated for each instance, and the best one is retained.

Tables 1, 2, 3 and 4 summarize the results. For each instance, problem parameters, computation time, and the percentage deviation, if not the optimality gap[2], are presented. Based on a detailed analysis of the results, we summarize the findings as follows:

- Our algorithm returns solutions for a big majority of the instances (241 out of 276). Among 241 solutions, 190 have either a percentage deviation or an optimality gap less than 1%.

[2]If the optimal solution is known, the deviation from it is reported. Otherwise, we report the optimality gap measured by the percentage deviation from the lower bound.

- Our algorithm is very efficient in finding good quality solutions for very large-sized instances (1,000 tasks) that correspond to real-life problems in a short time (Table 4 shows an average optimality gap of 0.81% and an average computational time of 184 seconds).

- Our algorithm finds very good solutions for medium- and large-sized instances very quickly, usually in less than one second (Tables 1, 2 and 3).

- Regarding the instances for which the optimal value of the objective function is known, our results present an average deviation of 0.47% (ranging from 0% to 1.52%, see Table 1) from this value. For the other medium- and large-sized instances, we present low optimality gaps with an average of 0.68% (from 0.01% to 7.13%, see Table 2) and 0.80% (from 0.01% to 3.94%, see Table 3).

Examining the cases for which the heuristic did not return feasible solutions, we find that corresponding instances are characterized by high line efficiency and relatively complex precedence relations. This makes finding feasible solutions for these instances harder, especially that the parameter selection we performed was focused on producing high-quality solutions rather than simply feasible ones. Notice that these cases constitute a minority of 35 out of 276 instances. We further tried to solve these instances by implementing a constructive heuristic using the 21 priority rules without the probabilistic function. However, this did not work.

Table 1: Results of the problems for which optimal solutions are known. (Scholl, 1999)

Filename	n	C	K	time (s)	% dev	Filename	n	C	K	time (s)	% dev
						Heskia	28	138	8	0.093	0.11
									9	0.094	0.55
Mitchell	21	14	8	0.078	0.14			216	5	0.093	0.18
			9	0.062	0.32				6	0.078	0.70
			10	0.063	0.89				7	0.078	0.52
		21	5	0.577	0.00			342	3	1.249	0.00
			6	0.078	0.11				4	0.078	0.59
			7	0.062	1.52				5	0.078	0.18
		39	3	0.062	0.05	Buxey	29	27	13	0.109	0.74
			4	0.047	0.29				14	0.109	0.58
			5	0.062	0.63				15	0.094	0.94
Roszieg	25	14	10	0.094	0.00			36	10	0.109	0.27
			11	0.094	0.14				11	0.094	0.73
			12	0.109	0.75				12	0.078	1.42
		18	8	0.078	0.51			54	7	0.093	0.55
			9	0.078	0.69				8	0.094	1.23
			10	0.078	0.13				9	0.078	0.87
		32	4	0.078	0.00	Tonge70	70	234	16	0.234	0.10
			5	0.062	0.70				17	0.234	0.27
			6	0.078	0.08				18	0.219	0.27

Table 2: Results of the medium and large size problems for which optimal solutions are not known. (Scholl, 1999)

Filename	n	C	K	time (s)	optimality gap (%)	Filename	n	C	K	time (s)	optimality gap (%)
Heskia	28	138	10	0.094	2.01			56	30	0.391	0.77
									32	0.328	0.49
									34	0.328	1.97
Warnecke	58	54	31	-	-	Arc83	83	3786	21	0.359	0.29
			33	0.328	1.26				23	0.344	0.50
			35	0.297	1.74				25	0.313	0.71
		71	23	-	-			5824	14	0.265	0.09
			25	0.250	0.69				15	0.250	0.19
			27	0.235	1.04				16	0.234	0.37
		111	14	-	-			10816	8	0.234	0.08
			15	0.187	0.15				9	0.219	0.04
			16	0.188	0.60				10	0.219	0.20
Tonge70	70	160	23	0.296	0.10	Lutz2	89	11	49	-	-
			25	0.266	0.48				52	0.672	1.76
			27	0.265	1.09				55	0.625	1.73
		527	7	0.203	0.05			16	31	-	-
			8	0.188	0.34				33	0.438	0.77
			9	0.172	0.29				35	0.437	1.55
Wee-mag	75	28	63	0.499	0.57			21	24	-	-
			67	0.438	2.48				26	0.375	0.55
			71	0.422	7.13				28	0.375	1.05
		39	60	0.718	1.66	Lutz3	89	75	23	-	-
			63	0.703	1.21				25	0.359	0.29
			67	0.672	2.84				27	0.344	0.68

Continued on next page

Table 2 – *Continued from previous page*

Filename	n	C	K	time (s)	optimality gap (%)	Filename	n	C	K	time (s)	optimality gap (%)
		97	18	-	-			17067	9	0.374	0.01
			19	0.328	0.27				10	0.328	0.16
			20	1.000	0.47				11	0.329	0.23
		150	12	0.296	0.30	Barthol2	148	84	51	-	-
			13	0.281	0.25				54	0.891	0.24
			14	0.297	0.28				57	0.812	0.53
Mukherje	94	176	25	0.437	0.15			112	38	-	-
			27	0.406	0.47				40	0.719	0.12
			29	0.390	1.07				42	0.671	0.32
		234	19	0.375	0.14			170	25	-	-
			20	0.359	0.41				27	0.578	0.09
			21	0.344	0.68				29	0.531	0.24
		351	13	0.328	0.09	Scholl	297	1394	51	1.265	0.03
			14	0.313	0.29				54	1.156	0.17
			15	0.312	0.54				57	1.109	0.34
Arc111	111	5755	27	0.515	0.19			1834	38	-	-
			29	0.515	0.45				40	1.077	0.12
			31	0.485	1.20				42	1.031	0.16
		7916	20	0.453	0.03			2787	25	-	-
			21	0.437	0.13				27	0.953	0.07
			23	0.422	0.96				29	0.906	0.14

Table 3: Results of the Large Size Problems with $n = 100$ tasks and $C = 1000$. (Otto et al., 2013)

			optimality				optimality				optimality
#file	K	time (s)	gap (%)	#file	K	time (s)	gap (%)	#file	K	time (s)	gap (%)
38	14	0.328	0.03	252	14	0.329	0.02	412	14	0.359	0.02
	15	0.297	0.20		15	0.328	0.10		15	0.343	0.18
	16	0.297	0.32		16	0.328	0.14		16	0.344	0.27
61	54	-	-	325	25	-	-	438	57	0.734	2.19
	57	-	-		27	0.422	0.29		60	0.703	2.76
	60	0.640	2.85		29	0.375	0.66		63	0.687	3.24
77	20	0.438	0.02	332	14	0.359	0.02	452	22	-	-
	21	0.406	0.07		15	0.313	0.12		24	0.406	0.32
	23	0.375	0.61		16	0.312	0.23		26	0.391	1.01
107	14	0.360	0.05	348	14	0.328	0.07	464	25	0.453	0.20
	15	0.343	0.26		15	0.312	0.12		27	0.469	0.30
	16	0.328	0.29		16	0.297	0.27		29	0.453	0.82
108	14	-	-	349	13	0.344	0.06	481	15	-	-
	15	0.390	0.10		14	0.312	0.10		16	0.390	0.14
	16	0.375	0.20		15	0.313	0.28		17	0.375	0.43
114	13	0.359	0.01	361	52	-	-	496	14	0.406	0.07
	14	0.344	0.12		55	0.656	1.76		15	0.406	0.17
	15	0.312	0.26		58	0.625	2.60		16	0.391	0.23
209	56	-	-	389	23	0.453	0.07	514	60	1.093	3.27
	59	0.656	3.44		25	0.422	0.26		63	1.062	3.48
	62	0.625	3.62		27	0.421	0.68		67	1.016	3.94
218	56	-	-	399	23	0.422	0.03				
	59	0.719	2.64		25	0.391	0.25				
	62	0.687	3.48		27	0.375	0.71				
244	21	0.391	0.08	400	24	-	-				
	23	0.391	0.30		26	0.453	0.14				
	25	0.359	0.46		28	0.406	0.62				

Table 4: Results of the Very Large Size Problems with $n = 1000$ tasks and $C = 1000$. (Otto et al., 2013)

			optimality				optimality				optimality
#file	K	time (s)	gap (%)	#file	K	time (s)	gap (%)	#file	K	time (s)	gap (%)
11	134	-	-	168	138	-	-	406	542	374.967	1.35
	141	68.423	0.08		145	58.487	0.09		570	353.621	2.08
	149	57.613	0.17		153	51.816	0.23		599	332.064	2.79
32	527	317.817	0.77	205	231	122.862	0.01	453	138	-	-
	554	350.887	1.51		243	85.011	0.11		145	126.487	0.09
	582	333.755	2.29		256	74.186	0.38		153	120.378	0.27
59	224	124.535	0.01	279	218	147.918	0.02	468	138	-	-
	236	99.478	0.14		229	72.452	0.12		145	117.785	0.12
	248	81.091	0.42		241	66.656	0.39		153	114.459	0.30
60	232	120.925	0.01	317	136	-	-	475	136	-	-
	244	97.665	0.13		143	67.937	0.09		143	119.738	0.08
	257	79.935	0.39		151	58.721	0.25		151	116.410	0.24
89	140	-	-	359	224	116.239	0.01	487	597	619.822	2.79
	147	66.565	0.10		236	94.572	0.12		627	605.187	3.16
	155	57.941	0.24		248	79.294	0.39		659	578.600	3.72
95	136	-	-	364	222	-	-	489	578	586.677	2.51
	143	63.548	0.10		234	95.431	0.13		607	562.337	2.88
	151	54.847	0.27		246	79.716	0.37		638	543.811	3.45
101	550	299.492	1.65	375	229	120.331	0.01	519	226	-	-
	578	276.716	2.05		241	96.993	0.13		238	149.059	0.18
	607	260.610	2.69		254	81.138	0.40		250	143.591	0.45
136	230	-	-	394	138	-	-				
	242	86.682	0.14		145	61.876	0.06				
	255	75.326	0.38		153	55.691	0.22				
144	220	98.024	0.01	403	557	402.442	1.68				
	231	78.997	0.14		585	380.226	2.40				
	243	68.390	0.40		615	361.339	3.06				

5. Conclusion

We develop an efficient heuristic to solve the workload smoothing problem in simple assembly lines by investigating and integrating the problem-specific characteristics. To build our algorithm, we incorporate 21 priority rules traditionally used in line balancing problems and a probabilistic decision-making procedure specifically designed to achieve the smoothing objective. To achieve the best integration strategy between the priority rules and the parameter of the probabilistic rule for closing stations α, we perform a comprehensive pilot test and select the best-performing combinations (rule $\times$ value of α). Based on this experimental design work, we select 72 combinations to be definitely integrated into our algorithm. These combinations result from nine values of α combined with eight priority rules as it could be seen in Appendix C.

We test the efficiency of our heuristic algorithm through an extensive computational experiment based on three data sets from Scholl (1999) and Otto et al. (2013). Analysis of the test results demonstrates the efficiency of our heuristic in producing high-quality solutions for medium-sized to very large-sized instances (up to 1,000 tasks) in a very short time.

Though we obtain excellent results regarding our problem, this work has its limits and can be further developed by exploring new promising research directions:

- Further investigation can be done to increase the percentage of feasible solutions generated by the heuristic. This includes the analysis of special cases of the problem, such as those that require a very high-capacity utilization and involve complex network structures.

- Our solution approach might be adapted for solving type III problems

in mixed-model lines.

- Lines with special flexibility and reconfigurability requirements can also be studied. In this regard, the study can be extended to develop heuristics for workload smoothing in U-type configurations.

- Multicriteria versions of the problem, such as a study of the line efficiency and/or robustness criteria in addition to the smoothing objective, can be explored.

References

Amen, M., 2000. Heuristic methods for cost-oriented assembly line balancing: A survey. International Journal of Production Economics 68, 1 – 14.

Azizoğlu, M., İmat, S., 2018. Workload smoothing in simple assembly line balancing. Computers & Operations Research 89, 51 – 57.

Battaïa, O., Dolgui, A., 2013. A taxonomy of line balancing problems and their solution approaches. International Journal of Production Economics 142, 259 – 277.

Boysen, N., Fliedner, M., Scholl, A., 2008. Assembly line balancing: Which model to use when? International Journal of Production Economics 111, 509–528.

Emde, S., Boysen, N., Scholl, A., 2010. Balancing mixed-model assembly lines: a computational evaluation of objectives to smoothen workload. International Journal of Production Research 48, 3173–3191.

Eswaramoorthi, M., Kathiresan, G., Jayasudhan, T., Prasad, P., Mohanram, P., 2012. Flow index based line balancing: a tool to improve the leanness

of assembly line design. International Journal of Production Research 50, 3345–3358.

Finco, S., Battini, D., Delorme, X., Persona, A., Sgarbossa, F., 2019. Workers' rest allowance and smoothing of the workload in assembly lines. International Journal of Production Research doi:10.1080/00207543.2019.1616847.

Groover, M.P., 2013. Work systems and the methods, measurement, and management of work. Pearson Prentice Hall Upper Saddle River, NJ.

Kim, Y.J., Kim, Y.K., Cho, Y., 1998. A heuristic-based genetic algorithm for workload smoothing in assembly lines. Computers & Operations Research 25, 99 – 111.

Kim, Y.K., Kim, Y.J., Kim, Y., 1996. Genetic algorithms for assembly line balancing with various objectives. Computers & Industrial Engineering 30, 397 – 409.

Li, Z., Kucukkoc, I., Nilakantan, J.M., 2017a. Comprehensive review and evaluation of heuristics and meta-heuristics for two-sided assembly line balancing problem. Computers & Operations Research 84, 146 – 161.

Li, Z., Tang, Q., Zhang, L., 2017b. Two-sided assembly line balancing problem of type i: Improvements, a simple algorithm and a comprehensive study. Computers & Operations Research 79, 78 – 93.

Moodie, C.L., Young, H.H., 1965. A heuristic method of assembly line balancing for assumptions of constant or variable work element times. Journal of Industrial Engineering 16, 23–29.

Mozdgir, A., Mahdavi, I., Badeleh, I.S., Solimanpur, M., 2013. Using the taguchi method to optimize the differential evolution algorithm parameters for minimizing the workload smoothness index in simple assembly line balancing. Mathematical and Computer Modelling 57, 137 – 151.

Nearchou, A.C., 2011. Maximizing production rate and workload smoothing in assembly lines using particle swarm optimization. International Journal of Production Economics 129, 242 – 250.

Otto, A., Otto, C., 2014. How to design effective priority rules: Example of simple assembly line balancing. Computers & Industrial Engineering 69, 43 – 52.

Otto, A., Otto, C., Scholl, A., 2013. Systematic data generation and test design for solution algorithms on the example of salbpgen for assembly line balancing. European Journal of Operational Research 228, 33 – 45.

Otto, A., Scholl, A., 2011. Incorporating ergonomic risks into assembly line balancing. European Journal of Operational Research 212, 277 – 286.

Pape, T., 2015. Heuristics and lower bounds for the simple assembly line balancing problem type 1: Overview, computational tests and improvements. European Journal of Operational Research 240, 32 – 42.

Pinnoi, A., Wilhelm, W.F., 1997. A branch and cut approach for workload smoothing on assembly lines. INFORMS Journal on Computing 9, 335–350.

Rachamadugu, R., Talbot, B., 1991. Improving the equality of workload assignments in assembly lines. International Journal of Production Research 29, 619–633.

Scholl, A., 1999. Balancing and Sequencing of Assembly Lines. 2 ed., Physica-Verlag Heidelberg.

Scholl, A., Becker, C., 2006. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. European Journal of Operational Research 168, 666–693.

Uğurdağ, H.F., Rachamadugu, R., Papachristou, C.A., 1997. Designing paced assembly lines with fixed number of stations. European Journal of Operational Research 102, 488 – 501.

Walter, R., 2020. A note on "workload smoothing in simple assembly line balancing". Computers & Operations Research 113, 104803.

Appendix A. Priority rules (Otto and Otto, 2014)

T: Task time

TdS: Task time divided by slack

TdL: Task time divided by latest station

PW^*: Positional weight

PW: Positional weight based on direct followers

APW: Average ranked positional weight

PW^v: Positional weight based on available followers

CPW: Recursive cumulated positional weight

CP: Critical path - maximum path processing time among all paths

L: Latest station

S: Slack

FdS: Total number of followers divided by slack

LdF: Latest station divided by number of followers

LP: Longest path - maximum number of followers in a path

$|F^*|$: Total number of followers

$|F|$: Number of direct followers

$|F^v|$: Number of available followers

$|F^s|$: Number of assignable followers

BNN: Number of bottleneck nodes within all followers

RE: Recursive cumulated edges

CF: Cumulated number of followers

Appendix B. Numerical example

In this numerical example, we consider a seven-task problem with the cycle time value $C = 12$ and a number of stations $K = 3$. Figure B.3 shows details on task times and precedence relations.

As mentioned in section 3.1, our algorithm generates several sets of solutions using predetermined combinations of priority rules and values of the parameter α. In this example, we use Longest Task Time and $\alpha = 0.6$ and explain how our algorithm builds a solution:

1. The order of the tasks according to the priority rule is: (C, D, E, G, A, B, and F).
2. Using formula (7) and (8), the threshold for station 1 is $T_1 = \frac{\sum_{i=A}^{G} t_i}{K} = \frac{3+2+7+6+5+1+4}{3} = \frac{28}{3}$, and the minimum closing load value in this station is $MCL_1 = \alpha T_1 = 0.6 \times \frac{28}{3} = 5.6$.
3. The set of eligible tasks is $\{A, B\}$.
4. Task A is assigned to station 1 as it has the highest priority.
5. To decide whether the station should be closed or not, we use (9) to compute the value $\mathbb{P}(\text{close } r)$. Here $\mathbb{P}(\text{close } r) = 0$ since $w_1 = 3 < 5.6 = MCL_1$.
6. A random uniform number is generated in $[0, 1]$ to be used for making the decision of closing the stations after each assignment. The number we obtain is 0.65.
7. As $0 < 0.65$, we decide to keep station 1 open and continue loading it.
8. The set of eligible tasks is updated, and the task with the highest priority in this set (task C) is identified and assigned.
9. As in step 5, we compute the new value of $\mathbb{P}(\text{close } r)$ and find $\mathbb{P}(\text{close } r) = \frac{(3+7)-5.6}{12-5.6} = 0.6875$.

10. As $0.6875 > 0.65$, the decision is to close station 1.
11. Station 2 is opened, and the same process is applied through which tasks B, D and F are assigned to this station. Observe that $\mathbb{P}(\text{close } r) = \dfrac{(2+6+1) - 0.6 \times 9}{12 - 5.4} = \dfrac{6}{11} < 0.65$. Thus, the decision resulting from the probabilistic rule (section 3.1.2) is not to close station 2. However, because none of the remaining tasks satisfy the constraints, station 2 is closed and station 3 is opened for assignment.
12. Following the same procedure, tasks E and G are assigned to station 3.

Using (6), the objective function value associated to this solution is $Z = 10^2 + 9^2 + 9^2 = 262$. Figure B.4 illustrates the feasible assignment of tasks to stations leading to this solution, wherein letters refer to the tasks, and numbers in parentheses refer to their processing times. Observe that:

- For the same combination of a priority rule and a α value, we may obtain different solutions simply if the random number generated in step 6 is different. That's why our algorithm generates a set of different solutions for each combination of a priority rule and a α value.

- Similarly, a different set of solutions can be obtained for any other combination of a priority rule and a α value.

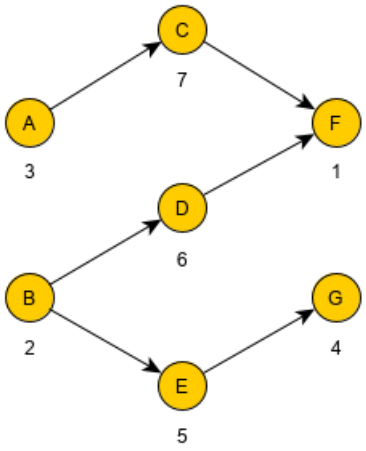

Figure B.3: Precedence network with processing times for the example in Appendix B.

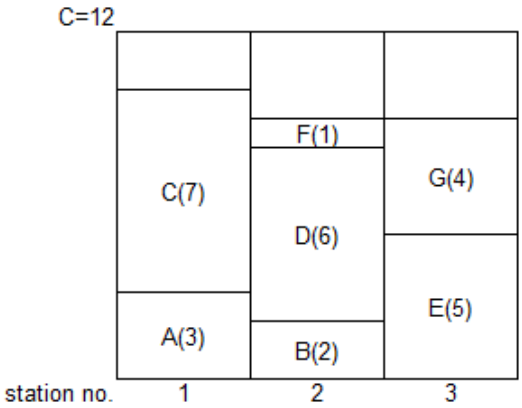

Figure B.4: A feasible solution obtained by our algorithm for the example in Appendix B.

Appendix C. Results of Pilot Tests on 39 medium-sized instances, sorted by increasing average percentage deviations.

rule	α	avg. %dev	rule	α	avg. %dev	rule	α	avg. %dev	rule	α	avg. %dev
CP	0.4	0.96%	CF	0.2	1.08%	CPW	0.2	1.18%	S	0	2.43%
CP	0.7	0.97%	RE	0.4	1.08%	CPW	0.1	1.18%	S	0.2	2.51%
CP	0.5	0.97%	CF	0.1	1.08%	CPW	0.8	1.19%	S	0.3	2.56%
CP	0.6	0.97%	LP	0.6	1.09%	LdF	0.3	1.20%	S	0.4	2.56%
CP	0.3	1.00%	RE	0.3	1.09%	$\|F^*\|$	0.3	1.21%	S	0.5	2.62%
CP	0.1	1.00%	LP	0.3	1.10%	$\|F^*\|$	0.4	1.21%	S	0.6	2.87%
CF	0.6	1.01%	RE	0.8	1.10%	LdF	0.2	1.21%	S	0.7	3.17%
RE	0.6	1.01%	LP	0.5	1.11%	$\|F^*\|$	0.6	1.21%	S	0.8	3.30%
CP	0.2	1.01%	CF	0	1.11%	CPW	0	1.22%	S	0.9	3.64%
PW^*	0.6	1.02%	CPW	0.5	1.12%	$\|F^*\|$	0.1	1.22%	BNN	0	5.56%
PW^*	0.5	1.02%	CPW	0.6	1.12%	LP	0	1.22%	BNN	0.3	5.64%
PW^*	0.7	1.02%	LdF	0.6	1.12%	$\|F^*\|$	0.5	1.22%	BNN	0.4	5.66%
PW^*	0.4	1.03%	RE	0.1	1.13%	$\|F^*\|$	0.2	1.22%	BNN	0.1	5.70%
CF	0.5	1.03%	RE	0.2	1.13%	LdF	0.1	1.23%	BNN	0.6	5.72%
RE	0.5	1.03%	LP	0.4	1.13%	LdF	0	1.23%	BNN	0.5	5.72%
CP	0	1.04%	LdF	0.7	1.13%	$\|F^*\|$	0.7	1.24%	BNN	0.2	5.75%
CF	0.7	1.05%	PW^*	0	1.13%	$\|F^*\|$	0	1.27%	APW	0.3	5.95%
CF	0.3	1.05%	RE	0	1.13%	$\|F^*\|$	0.8	1.27%	APW	0.2	6.01%
CP	0.8	1.05%	PW^*	0.8	1.13%	CP	0.9	1.51%	$\|F^s\|$	0.4	6.06%
PW^*	0.2	1.05%	CPW	0.7	1.13%	LP	0.9	1.62%	APW	0.4	6.33%
RE	0.7	1.06%	LdF	0.5	1.15%	PW^*	0.9	1.68%	$\|F^s\|$	0	6.33%
CF	0.4	1.07%	LdF	0.4	1.15%	$\|F^*\|$	0.9	1.75%	PW^v	0.5	6.34%
PW^*	0.1	1.07%	LP	0.2	1.15%	CPW	0.9	1.79%	L	0.4	6.36%
PW^*	0.3	1.07%	LdF	0.8	1.16%	RE	0.9	1.83%	PW^v	0.4	6.36%
LP	0.8	1.07%	LP	0.1	1.17%	CF	0.9	1.86%	$\|F^s\|$	0.1	6.37%
CF	0.8	1.08%	CPW	0.3	1.18%	LdF	0.9	1.97%	L	0.2	6.40%
LP	0.7	1.08%	CPW	0.4	1.18%	S	0.1	2.39%	BNN	0.7	6.42%

Continued on next page

Table C.5 – *Continued from previous page*

rule	α	avg. %dev	rule	α	avg. %dev	rule	α	avg. %dev	rule	α	avg. %dev
L	0	6.45%	TdL	0.2	6.96%	TdS	0.3	8.27%	$\|F^s\|$	0.8	9.43%
BNN	0.8	6.46%	FdS	0.5	7.03%	T	0.8	8.31%	TdL	0.8	9.51%
$\|F^v\|$	0.1	6.47%	FdS	0.1	7.18%	TdS	0	8.33%	TdL	0.9	9.52%
$\|F^s\|$	0.2	6.52%	T	0.1	7.21%	TdS	0.2	8.35%	PW	0.3	9.58%
T	0.2	6.53%	FdS	0.6	7.26%	TdS	0.4	8.35%	$\|F^v\|$	0.8	9.60%
$\|F^v\|$	0.2	6.59%	T	0.5	7.49%	FdS	0.9	8.38%	TdS	0.9	9.64%
T	0.3	6.60%	PW^v	0.1	7.60%	TdL	0	8.39%	$\|F^s\|$	0.9	9.64%
L	0.3	6.60%	FdS	0.4	7.63%	TdL	0.6	8.40%	$\|F^v\|$	0.7	9.66%
L	0.1	6.61%	FdS	0.2	7.69%	T	0.9	8.43%	PW	0.2	9.69%
L	0.5	6.64%	PW^v	0.6	7.70%	APW	0	8.51%	PW	0.1	9.80%
$\|F^v\|$	0	6.66%	TdL	0.5	7.73%	PW^v	0.8	8.54%	$\|F^v\|$	0.9	9.92%
$\|F^s\|$	0.3	6.68%	FdS	0.3	7.74%	TdS	0.1	8.55%	PW	0	9.99%
T	0.4	6.73%	L	0.7	7.81%	FdS	0	8.60%	$\|F\|$	0.3	10.11%
$\|F^v\|$	0.3	6.74%	TdS	0.8	7.83%	T	0	8.64%	PW	0.4	10.16%
$\|F^v\|$	0.4	6.74%	FdS	0.7	7.84%	$\|F\|$	0	8.70%	$\|F\|$	0.4	10.87%
BNN	0.9	6.74%	FdS	0.8	7.94%	TdS	0.5	8.71%	PW	0.5	11.02%
PW^v	0.3	6.80%	T	0.6	7.95%	PW^v	0.9	8.79%	$\|F\|$	0.5	11.18%
APW	0.5	6.82%	APW	0.6	7.95%	PW^v	0.7	8.79%	PW	0.6	11.55%
PW^v	0.2	6.83%	$\|F^s\|$	0.6	8.00%	TdL	0.7	8.90%	PW	0.7	12.04%
$\|F^s\|$	0.5	6.85%	L	0.8	8.10%	TdS	0.6	8.92%	PW	0.8	12.04%
L	0.6	6.85%	APW	0.7	8.11%	$\|F\|$	0.1	9.05%	PW	0.9	12.04%
TdL	0.1	6.85%	APW	0.8	8.11%	$\|F\|$	0.2	9.23%	$\|F\|$	0.6	12.08%
$\|F^v\|$	0.5	6.90%	APW	0.9	8.12%	PW^v	0	9.26%	$\|F\|$	0.7	12.34%
TdL	0.3	6.90%	$\|F^v\|$	0.6	8.17%	$\|F^s\|$	0.7	9.29%	$\|F\|$	0.9	12.56%
APW	0.1	6.92%	L	0.9	8.22%	TdS	0.7	9.31%	$\|F\|$	0.8	12.57%
TdL	0.4	6.93%	T	0.7	8.25%						