Jena Research Papers in Business and Economics

SALBPGen – A systematic data generator for (simple) assembly line balancing

Alena Otto, Christian Otto and Armin Scholl

05/2011

Jenaer Schriften zur Wirtschaftswissenschaft

Working and Discussion Paper Series
School of Economics and Business Administration
Friedrich-Schiller-University Jena

ISSN 1864-3108

Publisher:

Wirtschaftswissenschaftliche Fakultät
Friedrich-Schiller-Universität Jena
Carl-Zeiß-Str. 3, D-07743 Jena

www.jbe.uni-jena.de

Editor:

Prof. Dr. Hans-Walter Lorenz
h.w.lorenz@wiwi.uni-jena.de

Prof. Dr. Armin Scholl
armin.scholl@wiwi.uni-jena.de

www.jbe.uni-jena.de

SALBPGen – A systematic data generator for (simple) assembly line balancing

Alena Otto[a], Christian Otto[a] and Armin Scholl[a,*]

[a]Friedrich-Schiller-University of Jena, Chair of Management Science,
Carl-Zeiß-Straße, D-07743 Jena, Germany
*Corresponding author: phone: +49 3641 943171, e-mail:armin.scholl@uni-jena.de

Abstract

Assembly line balancing is a well-known and extensively researched decision problem which arises when assembly line production systems are designed and operated. A large variety of real-world problem variations and elaborate solution methods were developed and presented in the academic literature in the past 60 years. Nevertheless, computational experiments examining and comparing the performance of solution procedures were mostly based on very limited data sets unsystematically collected from the literature and from some real-world cases. In particular, the precedence graphs used as the basis of former tests are limited in number and characteristics. As a consequence, former performance analyses suffer from a lack of systematics and statistical evidence.

In this article, we propose SALPBGen, a new instance generator for the simple assembly line balancing problem (SALBP) which can be applied to any other assembly line balancing problem, too. It is able to systematically create instances with very diverse structures under full control of the experiment's designer. In particular, based on our analysis of real-world problems from automotive and related industries, typical substructures of the precedence graph like chains, bottlenecks and modules can be generated and combined as required based on a detailed analysis of graph structures and structure measures like the order strength.

We also present a collection of new challenging benchmark data sets which are suited for comprehensive statistical tests in comparative studies of solution methods for SALBP and generalized problems as well. Researchers are invited to participate in a challenge to solve these new problem instances.

Keywords: manufacturing; benchmark data set; assembly line balancing; precedence graph; structure analysis; complexity measures.

1 Introduction

The problem of an optimal allocation of tasks to working stations of an assembly line, or assembly line balancing problem (ALBP), describes the planning situation in manual assemblies. It can be found, for example, in automotive, white goods and electronics industries. Since its formulation by Salveson (1955), a considerable number of exact and heuristic methods for the ALBP, known to be NP-hard, were proposed and they are getting more and more effective.

Ongoing developments in research on ALBP set new requirements on benchmark data sets. A benchmark data set forms the basis for testing relative performance of computational methods. Hereto, researchers are interested not only whether their methods perform better or worse in the given benchmark data set, but also, in *what kind of instances* its performance is especially weak or strong. A set of *structure measures* was proposed in the literature to investigate performance of solution methods and estimate the complexity of the problem instance. A list of comparative computational studies of solution methods for ALBP that emphasize the structure measures of data instances is very long and includes, for example, Mastor (1970), Dar-El (1975), Johnson (1981), Wee and Magazine (1981), Talbot et al. (1986), Hoffmann (1992), Scholl (1999, Chapter 7), Amen (2001), Levitin et al. (2006), Urban and Chiang (2006), Andrés et al. (2008) and Gao et al. (2009). Moreover, a deeper knowledge on the interactions of the problem structure and algorithms may facilitate creation of even more effective and efficient solution methods.

Assembly line balancing is a routine planning task at factories. The knowledge, which solution method to select is extremely important for practitioners. Thus, in their assembly line balancing software "A~Line", Driscoll and Thilakawardana (2001) used structure measures for a guided selection of a suitable solution heuristic.

The currently existing data sets as well as present data set generation methods exhibit rather limited structures of the graphs and do not allow for a systematic variation of structure measures, such as described in the literature. Therefore, we propose a new data set generation method that overcomes this drawback and allows controlling for the most widely accepted structure measures.

We precede with a short introduction into ALBP and existing structure measures in Section 2. In Sections 3 and 4, we review existing data sets and data generation procedures, respectively. We describe our new data generation procedure SALBPGen in Section 5. In Section 6 we discuss the order strength structure measure and provide insights towards its connection to the optional parameters of SALBPGen. Further, characteristics of the instances generated by

SALBPGen (Section 7) and the new benchmark data set (Section 8) are described. Finally, we conclude with a summary and further research directions in Section 9.

2 Basics and structure measures of ALBP

Formally speaking, an assembly line is a production system, where a set of *tasks* $V = \{1, \dots, n\}$ with operation times t_i (for $i \in V$) are distributed among a set of (*work)stations* $W = 1, \dots, U$ arranged in a sequential order. The workpieces are launched down the line at a fixed rate such that each station has access to every workpiece for a constant timespan. Within this *cycle time* c, the worker operating at station $u \in W$ has to perform all the tasks contained in the *station load* $S_u \subset V$, i.e., the station time $t(S_u) = \sum_{i \in S_u} t_i$ must not exceed the cycle time c. This process is repeated for every new workpiece cyclically.

Due to technological and organizational conditions, the order of performing the tasks is restricted by *precedence relations* (i, j), meaning that a task $i \in V$ must be executed before another task $j \in V$. The production process can be summarized by a non-cyclical digraph $G = (V, E, t)$, called *precedence graph*, where V is the node set and $E = \{(i,j) | i \in V,\ j \in F_i\}$ is the set of arcs representing direct precedence relations (see Figure 1). F_i and P_i denote the set of direct followers and direct predecessors of task i, respectively. The transitive closure of E we call $E^T = \{(i,j) | i \in V,\ j \in F_i^T\}$ with F_i^T and P_i^T denoting the sets of indirect followers and predecessors of task i. The node weights represent the operation times t_i.

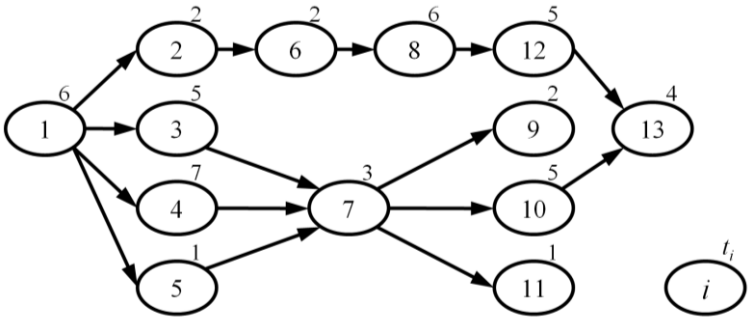


Figure 1: *Example of a precedence graph*

For the sake of convenience, we hypothetically insert a dummy source node with $t_0 = 0$ which is predecessor of all tasks $i \in \{1, \dots, n\}$. We define the *depth* of task i as the maximal number of arcs in a path connecting i with dummy source 0. In this terms, the stage $k \in \{1, \dots, K\}$ consists of all tasks of the precedence graph with depth k. For example, the precedence graph in Figure 1 has six stages: $\{1\}, \{2,3,4,5\}, \{6,7\}, \{8,9,10,11\}, \{12\}, \{13\}$.

In its basic form, the *Assembly Line Balancing Problem (ALBP)* is to assign tasks to stations such that cycle time and other restrictions as well as precedence relations are met and some time-, capacity-, cost- and/or profit-oriented goals are optimized (e.g. Amen 2001; Becker and Scholl 2006; Boysen et al. 2007). A feasible task assignment is called (line) balance (cf. Baybars 1986). A special case of ALBP is the *Simple Assembly Line Balancing Problem-1*

(SALBP-1), which is to minimize the number of workstations satisfying cycle time and precedence constraints (Scholl and Becker 2006). We restrict our further discussion to data instances and structure measures for SALBP-1, since it is the basic and most investigated version of ALBP. However, the results of the paper can be easily generalized to other variants of ALBP. For example, Scholl et al. (2008, 2009, 2010) as well as Otto and Scholl (2011) generated their testing instances based on an existing benchmark data set for SALBP-1.

Graph structure measures:	*Reference example from the literature*
Number of tasks: n	Elmagharaby and Herroelen (1980)
Order strength: $OS = \frac{2 \cdot \lvert E^T \rvert}{n \cdot (n-1)}$ (also called *TF*-ratio; and flexibility ratio $FR = 1 - OS$)	Mastor (1970), Dar-El (1975)
Average number of immediate predecessors: $AIP = \frac{\lvert E \rvert}{n}$	Rosenberg and Ziegler (1992)
Degree of divergence: $Div = 1 - \frac{\lvert E \rvert + s_1 - n}{ED}$, where s_1 is number of tasks without predecessors, $ED = \begin{cases} \lvert E \rvert + s_1, & if\ s_1 > 1 \\ \lvert E \rvert, & if\ s_1 = 1 \end{cases}$	Scholl (1999, Chapter 2.2.1.5)
Degree of convergence: $Conv = 1 - \frac{\lvert E \rvert + f - n}{EC}$, where f is number of tasks without successors, $EC = \begin{cases} \lvert E \rvert + f, & if\ f > 1 \\ \lvert E \rvert, & if\ f = 1 \end{cases}$	Scholl (1999, Chapter 2.2.1.5)
Presence of bottleneck tasks	Dar-El (1975)
Maximum task degree: $MD = \max_i\{\lvert P_i \rvert + \lvert F_i \rvert\}$	Baybars (1986)
Form and characteristics of stages	Scholl (1999, Chapter 2.2.1.5)
Number of tasks without predecessors: s_1	Gehrlein (1986)
Number of stages: K	Gehrlein (1986)
Number of feasible sequences	Elmagharaby and Herroelen (1980)
Time structure measures:	
Maxtime-ratio $\frac{t^{max}}{c}$, $t^{max} = \max_i\{t_i\}, i \in 1..n$ (and its inverse $\frac{c}{t^{max}}$)	Kilbridge and Wester (1961), Wee and Magazine (1981)
Mintime-ratio $\frac{c}{t^{min}}$, $t^{min} = \min_i\{t_i\}, i \in 1..n$	Wee and Magazine (1981)
Time variability ratio: $TV = \frac{t^{max}}{t^{min}}$	Scholl (1999, Chapter 2.2.1.5)
Sumtime-ratio: $c / \sum_{i=1}^{n} t_i$	Johnson (1981), Hoffmann (1990)
Standard deviation of task times: $std(t)$	Bhattacharjee and Sahu (1990)

Table 1: *Overview of structure measures for SALBP-1*

The structure measures proposed in the literature for comparative studies on solution methods of SALBP and as indicators for the instance's complexity are summarized in Table 1. They can be divided into: (1) *Graph structure measures* that describe the number and the precedence relations of tasks without connection to their task times. (2) *Time structure measures* that describe the properties of the set of task times (possibly in relation to the cycle time) independently from the tasks, to which they are assigned. (3) *Interaction measures* that

describe properties of the graph arising from assignment of certain task times to tasks with specific precedence relations to other tasks.

There has been done little systematical study of implications of structure measures for ALBP up to date. However, several results indicate that it is a promising area of research. In his studies of exact solution methods for SALBP-1, Johnson (1981) received that computational times decrease with the higher order strength of the precedence graph. Hoffmann (1992) for the exact solution method EUREKA and Talbot et al. (1986) for heuristic solution methods for SALBP-1 showed that higher maxtime-ratio increases computational times and deteriorates the performance of heuristic methods. The sumtime-ratio was first investigated by Hoffmann (1992). The results of his computational studies indicate that computational times of his exact solution method EUREKA increase, if the sum of task times is a multiple of the cycle time.

In more recent studies, researchers applied structure measures to investigate performance of solution methods for ALBP with additional extensions and constraints. As a rule, their findings support those found for SALBP. Investigations of u-shaped ALBP with stochastic task times by Urban and Chiang (2006) confirmed that higher order strength decreases computational times. In his study of robotic ALBP with objective to minimize the cycle time, Levitin (2006) received that performance of a genetic algorithm relative to a branch and bound method improved at lower order strength.

Also it was found that relative performance of different solution methods depends on the underlying structure of the problem. Thus, for example, Mastor (1970) in an analysis of SALBP-2 with objective to minimize the cycle time by an iterative solution of SALBP-1 found that the relative performance of heuristics, both measured in computational time and quality of solution, changes at different levels of OS and at different numbers of tasks. Talbot et al. (1986) in their study of heuristics for SALBP-1 found that no examined solution method consistently dominates the others, but their relative efficiency depends on the problem structure parameters.

A number of structure measures proposed in the literature are not extensively investigated yet. In his seminal article, Baybars (1986) wrote that measures of node connectivity, for example, *maximum degree* (or number of direct followers and predecessors of the task), could be better structure measures than the order strength. Inspired by Jackson (1956), Scholl (1999, Chapter 2.2.1.5) suggested that *form of stages, their precedence relations to neighboring stages* and *task times of their tasks* may be summarized into one/some easy to compute and informative structure measures. Based on the computational results of Talbot et al. (1986), Gehrlein (1986) noted, that further graph structure measures, such as *number of tasks without*

predecessors and *number of stages*, are important indicators for the (relative) performance of solution methods.

To our best knowledge, no structure measures that we classify as interaction ones, are proposed in the literature. Nevertheless they might be important. A prominent but little cited result received by Hoffmann (1990) is that interaction parameters matter a lot for the instance's complexity and for the (relative) performance of solution methods. Due to simple permutation of task times in Hoffmann's experiment, computational times of the examined solution method for some instances grew up by the factor of 210 for an instance containing 30 tasks.

3 Review of existing data sets

Many of the computational comparisons of solution methodologies are based on the benchmark data set of Scholl (1993) or its adaptations to the specific ALBP of interest (*www.assembly-line-balancing.de*). It is a combination and extension of the data sets of Talbot et al. (1986) and Hoffmann (1990, 1992) and contains 269 instances. They were collected from different sources, both generated and taken from empirical studies, based only on 25 precedence graphs (23 distinct graphs and two graphs with modified task times) with different cycle times. A detailed description of the benchmark data set with respect to the structure measures can be found in Scholl (1999, Chapter 7.1). Unfortunately, the data set does not allow for a systematic investigation of solution methods' performance at different levels of graph structure and interaction measures as well as for some time structure measures. For example, Table 2 shows that even for the structure parameters most established in the literature such as number of tasks and order strength, the number of precedence graphs per cell does not exceed 4, while others are empty. Further, only 20 precedence graphs (18 distinct graphs) have more than 25 tasks and thus are meaningful for comparing elaborate solution methods.

		Order strength		
		[0, 0.33]	*(0.33, 0.66]*	*(0.66, 1]*
Numberoftasks	*[0, 25)*	0	3	4
	[25, 50)	1	4	1
	[50, 100)	1	4	2+
	>100	0	3+	0

Table 2: *Benchmark data set of Scholl: Number of precedence graphs with certain properties ("+" marks graphs with the same structure but modified task times)*

Another and perhaps even a larger problem of the current benchmark data set is triviality of many contained problem instances. We call a problem instance *trivial*, if the share of optimal solutions in the feasible ones is (almost surely) 100%. For such a problem instance a simple random generation of one or a few feasible task sequence(s) would have a high probability to find an optimal solution. The lower this share of optimal solutions is, the more elaborate the solution method should be. Apparently, trivial

	c	(%)	c	(%)	c	(%)	c	(%)	c	(%)	c	(%)
Arcus 1	3786	0.3	3985	0.0	4206	0.0	4454	1.2	4732	0.4	5048	26.2
	5408	81.0	**5824**	96.2	**5853**	97.2	6309	87.8	6842	31.3	6883	59.9
	7571	100.0	**8412**	100.0	**8898**	99.9	**10816**	100.0				
Arcus 2	5755	0.0	5785	0.0	6016	0.0	6267	0.0	6540	0.0	6837	0.0
	7162	0.0	7520	0.0	7916	0.0	8356	3.7	8847	14.9	9400	36.1
	10027	19.5	10743	38.0	11378	58.2	**11570**	93.4	17067	40.5		
Bartholdi 1	705	0.0	No optimum found for other 7 instances									
Bartholdi 2	No optimum found for any of 27 instances											
Bowman	**20**	100.0										
Buxey	27	0.3	30	8.5	33	0.2	36	57.2	41	2.1	47	0.0
	54	99.2										
Gunther	41	0.8	44	4.4	49	30.7	54	0.0	61	51.1	69	62.7
	81	86.3										
Hahn	2004	51.6	2338	0.0	**2806**	100.0	**3507**	100.0	**4676**	100.0		
Heskiaoff	138	37.2	205	0.0	216	23.3	256	0.0	**324**	100.0	342	5.8
Jackson	**7**	100.0	**9**	100.0	10	10.7	13	85.0	14	9.1	21	70.8
Jaeschke	**6**	100.0	7	50.0	**8**	100.0	**10**	100.0	**18**	100.0		
Kilbrid	56	0.3	57	9.6	62	0.0	69	0.0	79	0.0	92	0.0
	110	100.0	111	4.8	138	0.4	184	1.6				
Lutz 1	1414	6.5	1572	0.3	1768	81.9	**2020**	96.4	**2357**	93.6	**2828**	100.0
Lutz 2	No optimum found for any of 11 instances											
Lutz 3	75	0.0	79	0.0	83	0.0	87	0.0	92	4.9	97	5.5
	103	50.1	110	0.0	118	0.0	127	57.0	**137**	99.7	**150**	95.8
Mansoor	48	25.0	62	9.1	94	50.0						
Mertens	6	22.2	7	60.0	**8**	100.0	**10**	100.0	15	28.6	18	44.4
Mitchell	14	0.0	15	11.6	21	2.3	**26**	100.0	35	1.4	**39**	100.0
Mukherjee	176	0.1	183	0.4	192	0.0	201	0.0	211	0.2	222	0.0
	234	6.9	248	27.9	263	70.9	281	88.4	301	53.3	**324**	92.2
	351	99.9										
Rosenberg	14	52.4	16	0.0	**18**	96.5	21	3.2	**25**	100.0	32	14.6
Sawyer	25	3.4	27	0.5	30	49.7	33	20.4	36	27.3	41	0.0
	47	0.0	**54**	99.9	**75**	100.0						
Scholl	No optimum found for any of 26 instances											
Tonge	160	0.1	168	0.0	176	0.0	185	0.3	195	0.4	207	5.6
	220	3.0	234	17.8	251	0.0	270	64.6	**293**	93.6	320	0.0
	364	55.1	**410**	91.7	**468**	96.0	**527**	91.4				
Warnecke	97	3.8	No optimum found for other 15 instances									
Wee-Mag	28	1.8	29	4.3	30	0.1	31	14.5	32	11.9	33	62.9
	34	95.6	35	14.6	36	85.3	**37**	99.4	**38**	100.0	**39**	100.0
	40	100.0	41	67.4	42	6.7	43	13.6	45	0.0	46	0.0
	47	0.0	49	0.1	50	4.1	52	0.1	54	31.7	56	0.2

Table 3: *Benchmark data set of Scholl: Share of optimal solutions in the solution space found by enumeration or a random search with 10,000 runs. Instances with share >90% are highlighted.*

problem instances are not suited to test or to compare (different) heuristic solution methods. Note, that it may still be hard to solve them with exact solution methods as those have to prove optimality.

To check, how many problem instances in the benchmark data set are trivial, we have taken 10,000 random solutions for each instance and computed the share of optimal solutions in them. For the small instances with less than 12 tasks, we made a full enumeration of feasible solutions, i.e., of feasible partitions of tasks between stations. In both cases, we applied the well-established maximum load rule which requires adding a further task to the current station load whenever it will not violate the cycle time constraint (Jackson 1956).

Table 3 summarizes the results by specifying the shares of optimal solutions in the feasible solution space. The lower this share is, the more challenging it will be to find an optimum. Of course, to find an exact share of optimal solutions for the NP-hard ALBP is not a realistic task. But its estimation in 10,000 random runs is sufficient for many purposes. The 95% confidence interval for the share of optimal solutions in the feasible solution space is at most ±0.01 for the shares near to 0.50 and less than ±0.006 for shares less than 0.001 or larger than 0.90 (see Appendix A.1). From Table 3, we see that for more than 57% of the instances an optimal solution was found by at least one of the 10,000 runs of our simple random search. Moreover, for 44 instances (16% of the data set), the share of optimal solutions in the solution space exceeds 90%. 24 problem instances appeared to be trivial, because all the solutions found in 10,000 runs of the random search were optimal.

Interestingly, neither a high number of tasks nor a low order strength guarantee a "non-triviality" of the problem instance. Among the trivial problem instances, we also find rather large ones with medium and low order strength, e.g., Arcus 1 with 83 tasks ($OS = 59\%$, $c = 10{,}816$) and Wee-Mag with 75 tasks ($OS = 23\%$, $c = 40$).

4 Review of existing data set generation methods

Due to restrictions of the available benchmark data set, a number of authors generated their own data sets varying selected structure measures of the data instances such as number of tasks, order strength and cycle times (e.g. Mastor 1970; Wee and Magazine 1981; Boctor 1995); Rubinovitz and Levitin 1995). Only three articles describe the used data generation procedure(s) explicitly: Gehrlein (1986), Bhattacharjee and Sahu (1990) and Rosenberg and Ziegler (1992).

In the generation methods of Gehrlein (1986) the order strength is a random parameter, which can be achieved with standard deviation of 1 to 11% depending on the size of the data set and the desired order strength. The method employs a two-step procedure: First, partial orderings of tasks are generated according to some rule (cf. Section 7). In the second step, a transitive closure of the task set is taken to calculate the order strength. No further structure parameters are controlled for.

In their random generator of precedence graphs, Bhattacharjee and Sahu (1990) actively use stages. They randomly generate the stages form, given the specified bounds on the number of stages, on the number of tasks per stage and on the number of direct followers for each task. However, they generate very specific graph structures, allowing direct precedence relations only between tasks on neighboring stages. This puts a significant limitation prohibiting construction of graphs typical in real-world settings. For example, in an assembly of a

pressure reducing device, described by Lutz (1974), about 20% of the tasks have at least one direct predecessor from a non-neighboring stage. Furthermore, in the generator of Bhattacharjee and Sahu, the target order strength and the desired number of tasks cannot be specified.

Instead of controlling for the order strength, Rosenberg and Ziegler (1992) set limitations on the range of immediate predecessors for each task. The generator cannot control for the order strength and further structure parameters.

There exist also a number of generators for related combinatorial optimization problems, but they are not directly suited for generating ALBP instances. For example, these are the generator of Arthur and Frednewey (1988) for the traveling salesman problem, the generator of Uyar and Uyar (2009) for the knapsack problem, ProGen (Kolisch et al. 1995), RanGen (Demeulemeester et al. 2003) and the generator of Browning and Yassine (2010) for the resource-constrained project scheduling problem, as well as generators for the nurse rostering problem (De Causmaecker and Berghe 2011).

Contrary to the existing generation methods for ALBP, our precedence graph generation procedure not only generates problem instances with the desired order strength, but also controls for further relevant parameters of the time and graph structure, such as number and characteristics of stages. Further, it enables to produce similar-to-real graphs by setting structure parameters to values typical for real-world assemblies.

5 SALBPGen – a new data generator for SALBP-1

The proposed new data generator SALBPGen is purposefully created for researchers, interested in examining solution methods. SALBPGen consists of two mandatory parts, the arc-generator and the task times-generator. Both parts can work independently of each other, but communication between them could be easily introduced in order to control for the interaction parameters. The user of SALBPGen may easily create instances to test effects of different task times for the same graphs, different graphs for the same set of task times and a simple permutation of task times for the same precedence graph. So, a systematic (statistical) analysis is supported by the configuration of the generator.

Also, the researcher can control for the following structure measures: number of tasks, order strength, number and characteristics of stages, distribution parameters of task times, the sumtime-ratio, the maximal number of direct followers and the number of *isolated* tasks (i.e. tasks without predecessors and successors).

The centerpiece of SALBPGen is the concept of stages. It allows for a direct manipulation of number of stages, stages form, precedence relations between stages and characteristics of task

times assigned to them, propagated by Gehrlein (1986) and Scholl (1999) as useful structure measures. Furthermore and even more important, the concept of stages is useful for controlling for other graph structures as well and create meaningful graphs even in terms of visual inspection.

For a higher resemblance with real-world precedence graphs, we enriched SALBPGen with the possibility of manipulating the number and form of bottleneck tasks, chains, as well as modules (Section 5.1). We proceed in Section 5.2 with the description of the basic algorithm of SALBPGen.

5.1 Bottlenecks, chains and modules

Bottleneck tasks are nodes with a high degree (number of direct followers and predecessors). We define a bottleneck as a task that is the only direct follower of at least two of its direct predecessors and the only direct predecessor of at least two of its direct followers. Hence, bottleneck tasks are at least of degree 4 with in-degree and out-degree, respectively, not smaller than 2. In Figure 1, task 7 is a bottleneck of degree 6 with direct predecessors $\{3, 4, 5\}$ and direct followers $\{9, 10, 11\}$.

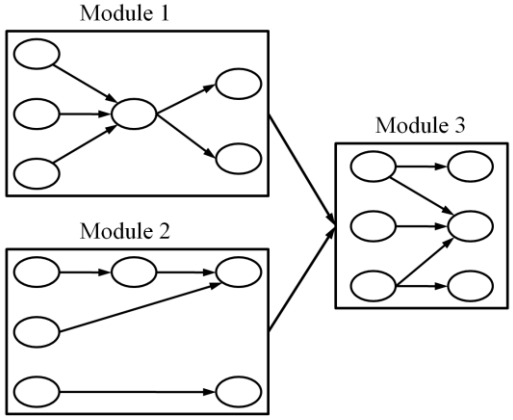
Chains are graph structures, whose constituent tasks make up a path and have at most one direct predecessor and at most one direct follower. In this paper, we speak about chains only in case they contain at least two tasks. In Figure 1, task 10 is not a chain tasks, because its direct predecessor has further direct successors and its direct successor has another direct predecessor. To the contrary, tasks 2, 6, 8 and 12 form a chain of length four; there are no further chains in this example.

Graph	***#tasks***	***OS***	***Share of tasks in chains (%)***	***Avg. chain length***	***Share of bottleneck tasks (%)***	***Avg. bottleneck degree***
Bartholdi	148	0.26	37%	2.6	3%	5.3
Gunther	35	0.60	43%	2.5	3%	5.0
Hahn	53	0.84	17%	2.3	2%	7.0
Heskiaoff	28	0.24	46%	2.2	–	–
Lutz1	32	0.84	44%	3.5	6%	6.0
Lutz2	89	0.78	33%	2.4	5%	5.5
Mukherje	94	0.45	3%	3.0	1%	14.0
Scholl	297	0.58	40%	2.8	3%	7.9
Tonge	70	0.59	36%	2.8	4%	8.0
Warnecke	58	0.59	33%	2.4	2%	6.0

Table 4: *Real-world precedence graphs from Scholl data set: frequency of chain tasks and bottleneck tasks*

From our cooperation with several firms in automotive industry, we gained insight that chains and bottleneck tasks are very common in real-world precedence graphs. A typical bottleneck task would be an examination task, such as examination of electric equipment in the central console: it immediately follows the mounting of electric equipment and must be completed before the carpeting will be placed, only then further tasks, as mounting of seats, can be performed. Clipping in of the airbag cable would be a typical chain task – a strictly predetermined sequence of tasks. Also our analysis of the *real-world* precedence graphs taken from the Scholl data set confirmed the frequent presence of these graph structures (see Table 4). In most cases, between 30 and 45% of the tasks are within chains, while 1 to 6% of the tasks are bottlenecks with an average degree between 5 and 14.

Complex products like automobiles are nowadays constructed in a *modular* design. This product structure is brought forward to the assembly production process, where, typically, the precedence graph is composed of subgraphs, each responsible for assembling such a module. This is done not only for technical constraints, but also in order to reduce organizational complexity and increase flexibility of production (Garud et al. 2003). Figure 2 shows a typical structure of a modular precedence graph, where precedence relations predominantly exist between modules (which can be seen as super nodes related to other super nodes in the module meta-graph). Examples of modules in automobile production would be wheel assembly, gas pedal module or trunk lining.



Matrix M $OS_{Mod} = \frac{2}{3}$		Modules		
		1	***2***	***3***
Modules	***1***	0	0	1
	2	0	0	1
	3	0	0	0

Figure 2: *Example of a precedence graph consisting of $m = 3$ modules, each forming a super node. The adjacency matrix M represents the transitive closure of the meta graph connecting the modules.*

5.2 Basic logic of SALBPGen

The basic algorithm of SALBPGen contains three parts: arc generation, task times generation and (if required) connection of single generated graphs as modules to form the final problem instance. Figure 3 summarizes required and optional parameters.

Arc generation. The user is required to specify the number of tasks and desired order strength OS with the tolerance range $\pm\delta^{OS}$. By default the tolerance parameter is set to $\delta^{OS} = 0.1$ for small graphs with $n \leq 20$ tasks, $\delta^{OS} = 0.05$ for medium graphs with $20 < n < 50$ tasks and to $\delta^{OS} = 0.005$ for large graphs with $n \geq 50$ tasks.

The user has an option either to apply a default random generator of stages, to set a rough direction for the stage form, for example, a diamond or a thin waist, or to specify exactly the number of tasks for each stage. The desired number of stages may be chosen by the user, whereby the default values are motivated by our investigations in Section 6.2 (see Table 5).

Arc generation	
Required parameters: number of tasks n and order strength OS	
Optional parameters / *default values*:	
Tolerance δ	$\pm 0.1, \pm 0.05$ *or* ± 0.005
(Exact) form of stages (incl. *K*)	*uniform (in expectation)*
Min No. of isolated tasks g	0
Max No. of direct followers for each task	n
Min No. of chains and distribution of their lengths	0
Min No. of bottlenecks and distribution of their degree	0
Task time generation	
Optional parameters / *default values*:	
Cycle time	1000
Distribution of task times	*Bimodal*
t^{max}, t^{min}	*not specified*
Sumtime-ratio must be integral	*not activated*
Module connector (optional)	
Required parameters: Super precedence matrix M for modules Reference to the settings of the constituent graphs (modules) specified in the arc and task time generation parts	

***Figure 3**: Constituent parts of SALBPGen*

For random variables, wherever needed, SALBPGen offers five kinds of distributions: uniform, normal, beta, bimodal (a mixture of two normal distributions) and exponential.

Optional parameters available in the arc generation part include the number of isolated tasks as well as the maximal number of direct followers of each task. Further, chain and bottleneck modes may be activated. In the chain mode, inputs are the number of chains and the distribution and its parameters according to which the length of chains will be generated. Similarly, in the bottleneck mode, the number of bottleneck tasks and the distributions for the number of direct predecessors and for the number of direct followers are set by the user. SALBPGen, if necessary, sets lower and upper bounds for the specified distributions

(truncation) in order to avoid contradictions with other parameters specified by users (for example, length of chains is restricted by the number of stages).

The arc-generation consists of the following four steps:

1. Lengths of chains as well as numbers of direct predecessors and followers for each bottleneck task are generated (according to the specified distributions) and their locations (stages) are defined.
2. The remaining tasks are distributed among the stages according to the stage-form parameters given by the user. If the stages form is not explicitly specified by the user, it is assumed to be uniform and SALBPGen randomly generates a number of tasks for each stage ensuring that the overall number of tasks fits the specified parameter n.
3. To assure that every task remains on its stage in the generated graph, we first connect each task j of stage $k > 1$ with exactly one (randomly chosen) task i of stage $k - 1$ paying attention to the maximal number of direct followers of task i.
4. We randomly insert non-redundant arcs and update the transitive closure of the graph until OS achieves the desired interval $[OS - \delta^{OS}, OS + \delta^{OS}]$ or no more non-redundant arcs are *allowed* due to constrictions of specified graph structures. A direct arc (i, j) is *allowed*, if:
   - the stage number of task i is smaller than that of j,
   - task i is not designated to be an isolated node,
   - restrictions for the specified bottlenecks, its direct predecessors and direct followers, given in the definition (see Section 5.1), are satisfied,
   - restrictions for the specified chains, given in the definition (see Section 5.1), are satisfied,
   - the maximal number of direct followers of task i will not be exceeded, and
   - by inserting arc (i, j) the OS value would not exceed the desired upper limit.

Task times generation. Here the user optionally sets the cycle time, minimal and maximal task times and characteristics of the distribution (chosen from the list given above), the task times are taken from. Following the proposition of Hoffmann (1992), it is also possible to generate task times, the sum of which is a multiple of the cycle time.

The task times are rounded to the next integer as some solution procedures require integer data. Possible rounding effects are compensated by setting default value of the cycle time to 1,000 as this is a matter of normalizing the time units only. Furthermore, this value seems to be large enough to flexibly generate a wide range for the time variability ratio and further time structure measures.

If the optional task time distribution parameters are not specified, SALBPGen selects the bimodal distribution, which is a two-step distribution. Firstly, we choose from which of two normal distributions the task time is to be drawn, then we generate a random number from the selected distribution. With probability 0.85, the lower normal distribution with expectation of $T_{avg} = 0.1 \cdot c$ and standard deviation of $0.15 \cdot c$ is selected. The upper normal distribution, chosen with probability 0.15, has expectation of $T_{avg} = 0.5 \cdot c$ and standard deviation of $0.05 \cdot c$. In each case the lower bound of task times is set to $0.02 \cdot c$ and the upper bound to c, i.e., the distributions are truncated correspondingly. The parameters are motivated by the empirical study of task time distributions at four assembly lines performed by Kilbridge and Wester (1961) and the task time distributions we experienced in different case studies in automotive industry.

Module connector (optional). Several graphs can be connected here to m modules $l \in \{1, \ldots, m\}$ according to a (super) precedence matrix M for modules specified by the user. Given the order strengths of the individual modules OS_l ($l \in \{1, \ldots, m\}$) and the order strength of module meta graph OS_M, SALBPGen simply connects the given subgraphs in the way specified by the meta graph adjacency matrix M and computes the final order strength of the resulting data instance as described in Section 6.1.

6 Implications of the parameters on the minimal and maximal order strength

The most widespread structure measure of ALBP instances is the order strength. This easy to compute measure also helps to approximate the number of feasible sequences, which is believed to indicate the complexity of problem instances (Thesen 1977; Elmagharaby and Herroelen 1980; Schwindt 1998). Besides the number of tasks, OS is the only required parameter for instance generation by SALBPGen. In the following, we sketch the implications of optional structure parameters of SALBPGen on the order strength of the resulting graph. We do it in order to illustrate, how and to which extent specifications of optional parameters restrict the range of possible graphs and which combinations are possible and useful.

6.1 Modules

There is an intimate relation of the order strengths of single modules OS_l, the order strength of the module matrix OS_{Mod} and the order strength OS of the resulting graph as the latter is the weighted sum of the first: $OS = \sum_{l=1}^{m} OS_l \cdot \omega_l + OS_{Mod} \cdot \omega_{Mod}$

Let m modules, with each module $l \in \{1, \ldots, m\}$ having order strength OS_l and containing τ_l tasks, are connected with precedence relations having order strength OS_{Mod}. Then, according to the definition of modules, order strength of the resulting graph will be:

$$OS = \sum_{l=1}^{m} OS_l \cdot \frac{\tau_l \cdot (\tau_l - 1)}{n \cdot (n-1)} + \frac{2 \cdot \sum_{i=1}^{m} \sum_{j=1}^{m} M_{ij} \cdot \tau_i \cdot \tau_j}{n \cdot (n-1)}$$

If all the modules have an equal number of tasks τ, i.e., $n = \tau \cdot m$, then:

$$OS = \sum_{l=1}^{m} OS_l \cdot \frac{\tau - 1}{m \cdot (\tau m - 1)} + OS_{Mod} \cdot \frac{\tau \cdot (m-1)}{\tau m - 1}$$

Example (see Figure 2): $m = 3$, $\tau = 6$, $OS_1 = \frac{11}{15}$, $OS_2 = \frac{5}{15}$, $OS_3 = \frac{5}{15}$ and $OS_{Mod} = \frac{2}{3}$. So, $\omega_{Mod} = \frac{12}{17}$ and $\omega_1 = \omega_2 = \omega_3 = \frac{5}{51}$. The resulting order strength is $\frac{21}{15} \cdot \frac{5}{51} + \frac{2}{3} \cdot \frac{12}{17} = \frac{31}{51}$.

Ceteris paribus, the influence of the module matrix, ω_{Mod}, on the order strength of the whole graph increases if:

- the number of modules increases,
- the size of modules is more uniformly distributed (maximal ω_{Mod} is achieved for modules of equal size),
- larger modules are subject to precedence relations.

If each module contains the same number of tasks, then the sum of weights equals to 1: $\omega_l = \omega, \forall l \epsilon \{1, \dots, m\}$ and $m \cdot \omega + \omega_{Mod} = 1$. Hereby for any number of tasks, the weight ω_{Mod} is more than 0.5 (if at least two modules are present) and it increases rapidly if the number of modules grows.

6.2 Stages

The predefined stages form restricts the possible value range of the order strength. Given the number of stages K and the number of tasks s_k for stage $k \in \{1, \dots, K\}$, a lower bound on the order strength is $OS_{min}(\boldsymbol{s}) = \frac{2}{n \cdot (n-1)} \sum_{k=2}^{K} s_k \cdot (k-1)$ as at least a path of $k-1$ tasks leads from a task on the first stage to each task at stage k. As an upper bound, we get $OS_{max}(\boldsymbol{s}) = \frac{2}{n \cdot (n-1)} \sum_{k=2}^{K} s_k \cdot \sum_{j=1}^{k-1} s_j$, because a task at stage k could be connected to all tasks at preceding stages directly or indirectly (see also Appendix A.2).

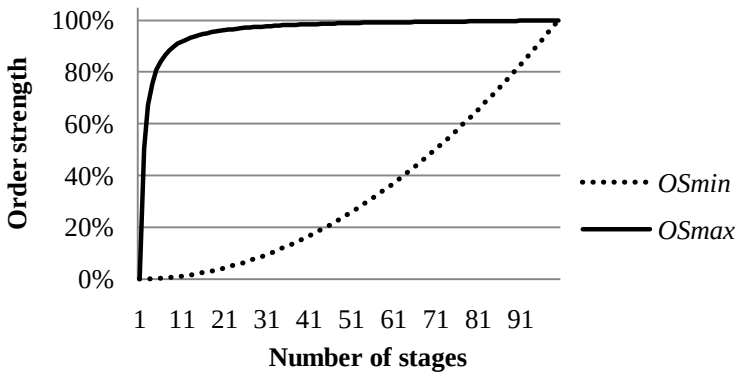


Figure 4: *Dynamics of OS_{min} and OS_{max} (with $n = 100$)*

Since the user of SALBPGen not always specifies the exact distribution of tasks to stages, we are also interested in defining the possible range of the order strength, if only the number K of

stages is given. Theorem 1 shows how to compute those values. Figure 4 visualizes the OS boundaries for an example instance with $n = 100$ tasks depending on the number K of stages.

The difference of both values can be interpreted as *precedence flexibility*, as it indicates the degree of flexibility to generate different precedence settings for a given number of stages. As a principle in problem generation, this flexibility should be as large as possible in order to be able to generate a large variety of possible precedence graphs. So, as a default, we set the number of stages to the value which guarantees largest precedence flexibility computed as given in Theorem 1. For the above example in Figure 4, about 17 stages, i.e., 6 tasks per stage on average, provide largest flexibility.

Theorem 1. Given the number of tasks n and the number of stages K or, equivalently, given the average number of tasks per stage n/K, the achievable order strength is bounded by $OS_{min} = \frac{K \cdot (K-1)}{n \cdot (n-1)}$ and $OS_{max} = \frac{\left(\binom{\theta}{2} \cdot \lceil \frac{n}{K} \rceil + \binom{K-\theta}{2} \cdot \lfloor \frac{n}{K} \rfloor + (K-\theta) \cdot \lfloor \frac{n}{K} \rfloor \cdot \theta \cdot \lceil \frac{n}{K} \rceil\right)}{\binom{n}{2}}$, where $\theta = n - \left\lfloor \frac{n}{K} \right\rfloor \cdot K$. The maximal precedence flexibility $OS_{max} - OS_{min}$ is observed at $n^2 = K^2 \cdot (2K - 1)$.

Proof: The main idea is that OS_{min} is lower, if there are more tasks on the first stages and to the contrary, OS_{max} is higher at a more uniform distribution of tasks by stages. For a detailed proof see Appendix A.3.

Table 5 shows the number of stages K for different problem sizes n to achieve the maximal flexibility. When generating instances with not extremely low or high levels of OS, these values are taken by default.

n	***20***	***30***	***40***	***50***	***60***	***70***	***80***	***90***	***100***	***125***	***150***	***175***	***200***	***300***
# stages	6	8	9	11	12	14	15	16	17	20	23	25	27	36

Table 5: *Number of stages, where the maximum flexibility in order strength is achieved*

6.3 Bottlenecks, chains and isolated tasks

The predefined form of stages has also an influence on the graph structure. For example, several stages with cardinality 1 in a row make emergence of chains more probable at low and high order strengths. Bottlenecks with high node degree are more likely to appear in graphs, where a stage with low and a stage with high cardinality follow each other.

Generally, presence of chains and bottleneck tasks increases the minimal achievable order strength and decreases the maximal one as the flexibility of the graph structure is reduced.

For a *bottleneck* assigned to some stage k $(K > k > 1)$ with $1 < p \leq \sum_{k'=1}^{k-1}(s_{k'} - 1)$ claimed (user-specified) direct predecessors and $1 < f \leq s_{k+1}$ claimed direct followers for given $s_1, \dots, s_K$ the minimal possible order strength is

$$OS_{min} = \frac{2}{n\cdot(n-1)} \cdot \left[\sum_{k'=2}^{K} s_{k'} \cdot (k'-1)) + (p-1)(f+1)\right],$$

because p claimed direct predecessors of the bottleneck lead to at least $p-1$ additional arcs for the bottleneck node and each of its claimed followers. The maximal possible order strength is achieved if all predecessors of the bottleneck are placed on the stage $k-1$, otherwise, among others, indirect arcs from further stages to the bottleneck node would be lost. Then, OS cannot exceed

$$OS_{max} = \frac{2}{n\cdot(n-1)} \cdot \left[\sum_{k'=2}^{K} s_{k'} \cdot \sum_{j=1}^{k'-1} s_j - (s_k - 1) \cdot (p+f)\right],$$

because p predecessors of the bottleneck task cannot have arcs to the other tasks on stage *k*. As well, all claimed followers cannot have predecessors on stage k except the bottleneck task.

Presence of a *chain* with a given length l that starts from stage t does not change the minimal achievable order strength.Technically assuming $s_0 = 1$ and $s_{K+1} = 1$, the chain restricts the maximal achievable order strength to

$$OS_{max} = \frac{2}{n\cdot(n-1)} \left[\sum_{k=2}^{K} s_k \cdot \sum_{j=1}^{k-1} s_j - \sum_{j=\mathrm{t}}^{t+l-1} (s_j - 1)(l-1) - (s_{t-1} + s_{t+l} - 2) \cdot l\right],$$

because along the whole length of the chain, other tasks at the stages cannot be connected with the chain tasks. Also, chain tasks are connected with only one task on stages adjacent to the chain. So the minimal restriction of the flexibility of the order strength generation will be, if there are no other tasks at the stages covered by the chain.

Given number of stages K, the number of tasks per stage $s_1, \dots, s_K$ and the number of *isolated tasks* $g < s_1$, the order strength cannot exceed

$$OS_{max} = \frac{2}{n\cdot(n-1)} \cdot \left(\sum_{k=3}^{K} s_k \cdot \sum_{k'=2}^{k-1} s_{k'} + \sum_{k=2}^{K} s_k \cdot (\mathrm{s}_1 - \mathrm{g})\right),$$

whereas the minimum possible order strength remains unchanged.

Maximal number of direct followers per task. Introducing this restriction, we followed the tradition of the generators of Bhattacharjee and Sahu (1990) and Rosenberg and Ziegler (1992), which contain limits on the number of direct predecessors or direct followers. The main role of this parameter is to restrict appearance of certain graph structures. There is no general direction of influence of this parameter neither on the minimal nor on the maximal achievable order strength.

7 Characteristics of instances generated by SALBPGen

Due to the stage generation concept, graphs generated by SALBPGen get a meaningful layout and structure even in default regime without external control for the structure parameters (see Figure 5).

To check the characteristics of the generated instances, we created a data set in the default modus of SALBPGen in a full-factorial manner for different numbers of tasks per instance $(20, 30, 40, 50, 80, 100, 1000)$ and various levels of the order strength ($OS = 0.1, 0.2, \dots, 0.9$) with 500 instances per cell. So, the data set for this experiment contains 31,500 instances.

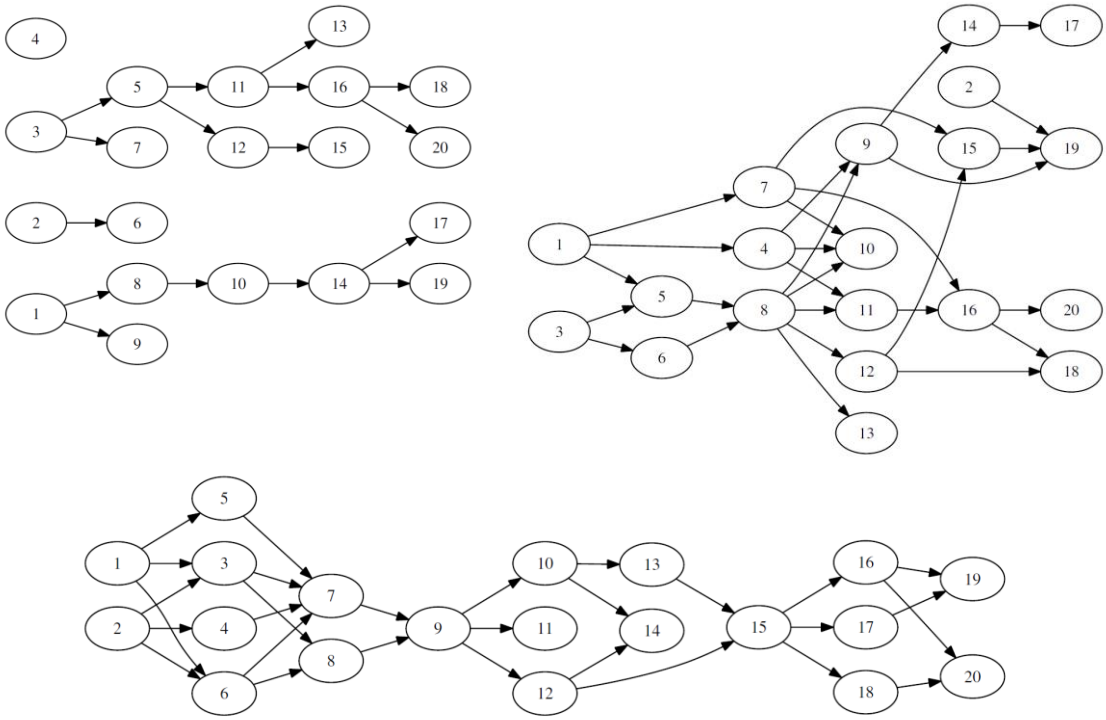


***Figure 5**: Three example graphs with different order strengths and default settings*

An important characteristic of a generator is its likelihood to generate different graphs, whereby we examine the dissimilarity of the graph structure and do not consider the assignment of task times. Among the instances generated by SALBPGen, no duplicate graphs were found for instances with more than 40 tasks. The share of duplicate graphs is negligible for generations of 20-task and 30-task instances (0.07% and 0.02%) respectively. Surely, for very high and very low levels of the order strength, the probability to receive duplicate graphs increases sharply. For example, independently on the number of tasks in the instance, only one distinctive graph (layout) is possible for $OS = 1$ and $OS = 0$, respectively.

For the examination whether the target order strength was achieved, we used tolerance intervals (δ^{OS}) as described in Section 5.2. The target order strength was always achieved for problem instances with $n \geq 40$ and for medium problem instances with $n = [20, 30]$ except for $OS = 0.1$. For $OS = 0.1$, the target order strength was achieved in 96 % for 30-task instances and in 57 % for 20-task instances. Overall, it is difficult to meet low levels of order strength for small problem instances.

Former generators (see Section 4) have difficulties in achieving the desired order strength even for medium and large instances. We illustrate it for 50-task problem instances. The generation was performed with the parameter values recommended by the authors of the respective method. For our suggested precision parameter of ±0.005, just 6–13% of the graphs generated by the method 1 of Gehrlein (1986), 16–47 % by his method 2 and 7–20% by his method 3 satisfied the required precision. The generator of Rosenberg and Ziegler (1992) resulted in a very large variability of order strengths at different parameter levels recommended by the authors and, thus, is not able to generate instances with desired order strength reliably. Furthermore, it is very unlikely to produce instances with small order strengths at all.

Each generator may occasionally create a precedence graph of any structure, but still has its own specificity. The methods of Gehrlein (especially method 2) are very unlikely to produce either chains or bottlenecks, default parameters of SALBPGen lead to a creation of some instances (up to 20%) containing chains and the generator of Rosenberg and Ziegler mostly(more than 80%) produces instances with chains and bottlenecks. We omit the method of Bhattacharjee and Sahu (1990), since it is very restrictive as it only produces graphs with a limited structure (see the discussion in Section 4).

However, in order to control for the minimum share of chain tasks and the number and characteristics of bottlenecks, we recommend to apply SALBPGen with specifying the modes "chains" and "bottlenecks". It is especially important for creating problem instances that show typical properties of real-world assembly processes (see Section 5.1).

8 New benchmark data set

We propose a collection of new diversified and challenging benchmark data sets. The new benchmark data sets are constructed by a full-factorial design for the following parameters: number of tasks ("small" with $n = 20$, "medium" with $n = 50$, "large" with $n = 100$ and "very large" with $n = 1000$), order strength ("low" $OS = 0.20$, "medium" $OS = 0.60$ and "high" $OS = 0.90$), distribution of task times, and type of the graph. It contains 25 observations per cell. Table 6 shows all useful combinations of parameters that sum up to 84 cells, i.e., 2,100 instances in total and 525 instances per graph size. We suggest to utilize the "medium" data (sub)set for testing exact solution methods, the "large" one is well suited for testing heuristics. We also include a "very large" data set since problems of this size are common in practice. The "small" data set is provided for speedy pre-tests of solution methods.

We employ three types of distributions for the task times: normal with peak at small tasks ("peak at the bottom"), bimodal, which is a combination of two normal distributions with a higher peak at small tasks (see Section 5.2), and normal with peak at $0.5 \cdot c$ ("peak in the middle"). The first two were motivated by the task time distributions typically found in the real-world, the third one is likely to constitute "hard" instances. This type of task time distribution is similar to that of the problem instances of Wee and Magazine contained in the Scholl benchmark data set, which contain many "hard" ones.

Precedence graphs are grouped into three categories according to the presence of chains and bottleneck tasks in them. Instances of the first one, "CH", contain at least 40% chain tasks. The second category, "BN", consists of instances containing bottleneck tasks with the least degree of eight except for small instances which observe a minimum degree of four. Category "MIXED" has no such structure requirements.

		Distribution of task times					
Graph structure	OS	„peak at the bottom“	„bimodal“	„peak in the middle“	„peak at the bottom“	„bimodal“	„peak in the middle“
		n = 20			*n = 50*		
MIXED	0.20	0/1/2/22/0	0/0/8/17/0	3/8/11/3/0	1/0/1/23/0	6/1/2/16/0	12/0/6/0/**7**
	0.60	0/2/7/16/0	0/3/2/20/0	4/7/12/2/0	4/0/4/17/0	9/4/8/4/0	14/0/0/0/**11**
	0.90	0/1/7/17/0	1/1/8/15/0	0/3/14/7/**1**	1/3/6/15/0	4/3/10/8/0	7/0/0/0/**18**
CH	0.20	0/0/8/17/0	1/1/7/16/0	4/9/10/2/0	5/0/2/18/0	8/3/8/6/0	16/1/0/0/**8**
	0.60	0/3/6/16/0	1/0/8/16/0	2/6/14/2/**1**	4/2/5/14/0	7/7/8/3/0	11/0/0/0/**14**
BN	0.20	0/3/5/17/0	1/1/4/19/0	2/13/7/3/0	0/1/3/21/0	6/5/7/7/0	20/0/0/0/**5**
	0.60	1/0/9/15/0	0/4/6/15/0	7/5/9/2/**2**	3/1/5/16/0	11/3/7/4/0	15/2/0/0/**8**
		n = 100			*n = 1000*		
MIXED	0.20	6/3/4/12/0	17/5/3/0/0	25/0/0/0/0	25/0/0/0/0	25/0/0/0/0	25/0/0/0/0
	0.60	6/5/3/11/0	18/5/1/0/1	17/0/0/0/**8**	25/0/0/0/0	25/0/0/0/0	25/0/0/0/0
	0.90	5/1/6/13/0	15/7/3/0/0	13/0/0/0/**12**	25/0/0/0/0	25/0/0/0/0	25/0/0/0/0
CH	0.20	5/1/6/13/0	13/5/5/0/**2**	22/0/0/0/**3**	25/0/0/0/0	25/0/0/0/0	25/0/0/0/0
	0.60	7/1/5/12/0	21/1/1/0/**2**	11/0/0/0/**14**	25/0/0/0/0	25/0/0/0/0	25/0/0/0/0
BN	0.20	6/1/4/14/0	15/3/6/1/0	23/0/0/0/**2**	25/0/0/0/0	25/0/0/0/0	25/0/0/0/0
	0.60	5/2/3/15/0	21/3/0/0/**1**	24/0/0/0/**1**	25/0/0/0/0	25/0/0/0/0	25/0/0/0/0

Table 6: *Trickiness of the instances in the new data set*
(No of "extremely tricky"/ "very tricky"/ "tricky"/ "less tricky"/"open")

We also report a *trickiness measure* Tr for each instance. We define the trickiness as a share of *non*-optimal solutions found by 10,000 runs of a random search method including the maximum load rule (see Section 3) in the solution space. It can be readily used as a reference of performance for heuristic solution methods. For the simplicity of use, we subdivide the instances into "extremely tricky" ($Tr \geq 0.995$), "very tricky" ($Tr \in [0.95, 0.995)$), "tricky" ($Tr \in [0.5, 0.95)$) and "less tricky" ($Tr \in (0, 0.5)$). For some unsolved instances, where during the random search the upper bound on the number of stations was found, the trickiness

measure remains "open". Note that this classification might be a bit misleading for exact solution methods, which also have to prove optimality of the found solution. We excluded the trivial instances, where all the solutions found by the random search were optimal ($Tr = 0$), from the benchmark data set, since such problem instances are badly suited for testing (heuristic) solution methods. A summary of the trickiness distributions can be found in Table 6. From the table, we can see that even the small benchmark data set with 20 tasks contains 27 (of 225) extremely tricky instances.

Note that the trickiness measure loses its power of differentiation for very large instances. Due to the size of these instances, the share of optimal solutions even for lighter instances is extremely low so that an immense increase of the number of runs for the random search would be required to capture it.

For the case, researchers would like to test thoroughly the role of the structure measures in their study of interest, we provide a further data set including permutations of medium instances ("medium permuted"). For each instance with $n = 50$ from the "medium" data set, nine instances with randomly permuted assignment of task times are added, i.e., these ten instances do not differ from each other in any possible graph structure and time structure measures. In some cases, a simple permutation of the task times may significantly change the complexity of the problem instance. For example, we found instances where one permutation shows a trickiness of $Tr > 0.9999$, whereas another one has a value $Tr < 0.0001$. We see it as an indication that working out of interaction measures (see Section 2) may be important for some research and real-world inquiries.

Besides being constructed in a systematic manner, the new benchmark data set is challenging. Contrary to the current benchmark data set (see Table 3), it does not contain trivial instances. A quarter of the new benchmark data set represents extremely tricky instances (see Table 6).

To find first optimal solutions for instances of the new benchmark data set, we run the exact solution procedure SALOME (Scholl and Klein 1997) with a low time limit for each instance (20, 50, 70 and 100 seconds, respectively, for the small, medium, large and very large data set). These preliminary results are summarized in Table 7. About 29% of the instances in the collection of new benchmark data sets are currently unsolved, especially large and very large instances. We welcome contributions to a competition on www.assembly-line-balancing.de for solving the

Data set	Total number of instances	Number of unsolved instances
small ($n = 20$)	525	4
medium ($n = 50$)	525	99
large ($n = 100$)	525	170
very large ($n = 1{,}000$)	525	339

***Table 7**: Current status for the known optimal solution of the new benchmark data set*

instances of the entire data set. The collection of new benchmark data sets as well as its detailed description can be downloaded from this site.

9 Summary and conclusions

Systematically generated data instances are required for performing comparative studies of solution methods. SALBPGen allows for a more thorough quality of computational experiments as it enables to generate sufficiently large data sets with systematically varied structure measures. To control the received results, SALBPGen enables a check for permuted task times, the same graph structures with different task times or the same sets of task times for different graphs.

The final goal of each investigation in operations research is to derive conclusions for real-world applications. Therefore, SALBPGen gives the possibility to generate data instances with structures that are typical for real-world assembly lines. Although it was created for SALBP-1, with few modifications this generator can be used for most variations of assembly line balancing problems as well as for some related scheduling and packing problems. (Boysen et al. 2008)

Systematically generated data instances are required for comparative studies of solution methods, but they are also needed to study further research questions. Since ALBP is NP-hard, exploiting knowledge on its structure is extremely valuable. For example, we could gain insights whether we could approximate the optimal ALBP solution by solving a less hard problem with a slightly modified precedence graph (cf. Klindworth et al. 2010). Further, with the very large data set, solution methods for more general ALBP (e.g. Becker and Scholl 2009; Scholl and Boysen 2009; Scholl et al. 2010) can be pre-tested before applying them in manufacturing planning systems.

Further research steps should include a systematic review of established as well as suggested and not yet stated structure measures on their predictive power of the problem's complexity and relative performance of solution methods. This study would not only revisit conclusions of previous inquiries, which were sometimes lacking statistical significance due to limited diversity of the tested data set. It would also aim at creating a "which method to use when?"-map as a guidance for practitioners and further research.

Acknowledgements

This article was supported by the Federal Program "ProExzellenz" of the Free State of Thuringia.

References

Amen, M. (2001). Heuristic methods for cost-oriented assembly line balancing: A comparison on solution quality and computing time. *International Journal of Production Research, 69*(3), 255–264.

Andrés, C., Miralles, C., & Pastor, R. (2008). Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. *European Journal of Operational Research, 187*(3), 1212–1223.

Arthur, J. L., & Frendewey, J. O. (1988). Generating travelling-salesman problems with known optimal tours. *The Journal of the Operational Research Society, 39*(2), 153–159.

Baybars, I. (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science, 32*(8), 909–932.

Becker, C., & Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research, 168*(3), 694–715.

Becker, C., & Scholl, A. (2009). Balancing assembly lines with variable parallel workplaces: Problem definition and effective solution procedure. *European Journal of Operational Research, 199*(2), 359–374.

Bhattacharjee, T., & Sahu, S. (1990). Complexity of single model assembly line balancing problems. *Engineering Costs and Production Economics, 18*(3), 203–214.

Boctor, F.F. (1995). A multiple-rule heuristic for assembly line balancing. *Journal of the Operational Research Society, 46*(1), 62–69.

Boysen, N.; Fliedner, & M.; Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research, 183*(2), 674–693.

Boysen, N., Fliedner, M., & Scholl, A. (2008). Assembly line balancing: Which model to use when? *International Journal of Production Economics, 111*(2), 509–528.

Brown, L. D., Cai, T. T., & DasGupta, A. (2001). Interval Estimation for a Binomial Proportion. *Statistical Science, 16*(2), 101–133.

Browning, T. R., & Yassine, A. A. (2010). A random generator of resource-constrained multi-project network problems. *Journal of Scheduling, 13*(2), 143–161.

Dar-El (Mansoor), E. M. (1975). Solving large single-model assembly line balancing problems – A comparative study. *IIE Transactions, 7*(3), 302–310.

De Causmaecker, P., & Berghe, G. V. (2011). A categorisation of nurse rostering problems. *Journal of Scheduling, 14*(1), 3–16.

Demeulemeester, E., Vanhoucke, M., & Herroelen, W. (2003). RanGen: A random network generator for activitity-on-the-node networks. *Journal of Scheduling, 6*(1), 17–38.

Driscoll, J., & Thilakawardana, D. (2001). The definition of assembly line balancing difficulty and evaluation of balance solution quality. *Robotics and Computer Integrated Manufacturing, 17*(1–2), 81–86.

Elmaghraby, S. E., & Herroelen, W. S. (1980). On the measurement of complexity in activity networks. *European Journal of Operational Research, 5*(4), 223–234.

Gao, J., Sun, L., Wang, L., & Gen, M. (2009). An efficient approach for type II robotic assembly line balancing problems. *Computers & Industrial Engineering, 56*(3), 1065–1080.

Garud, R., Kumaraswany, A, & Langlois, R. L. (2003). *Managing in the modular age: architectures, networks, and organizations.* Blackwell Publishers Ltd.

Gehrlein, W. V. (1986). On methods for generating random partial orders. *Operations Research Letters, 5*(6), 285–291.

Hoffmann, T. R. (1990). Assembly line balancing: a set of challenging problems. *International Journal of Production Research, 28*(10), 1807–1815.

Hoffmann, T. R. (1992). EUREKA – a hybrid system for assembly line balancing. *Management Science, 38*(1), 39–46.

Jackson, J.R. (1956). A computing procedure for a line balancing problem. *Management Science, 2*(3), 261–271.

Johnson, R. V. (1981). Assembly line balancing algorithms: Computational comparisons. *International Journal of Production Research, 19*(3), 277–287.

Kilbridge, M., &Wester, L. (1961). The balance delay problem. *Management science, 8*(1), 69–84.

Klindworth, H., Otto., C., & Scholl, A. (2010). On the learning precedence graph concept for the automotive industry. In: *Jena Research Papers in Business and Economics 9/2010*. Jena: School of Economics and Business Administration, Friedrich-Schiller University Jena.

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. *Management Science, 41*(10), 1693–1703.

Levitin, G., Rubinovitz, J., & Shnits, B. (2006). A genetic algorithm for robotic assembly line balancing. *European Journal of Operational Research, 168*(3), 811–825.

Lutz, L. (1974). *Abtakten von Montagelinien*. Mainz: Krausskopf.

Mastor, A. A. (1970). An experimental investigation and comparative evaluation of production line balancing techniques. *Management Science, 16*(11), 728–746.

Otto, A., & Scholl, A. (2011). Incorporating ergonomic risks into assembly line balancing. *European Journal of Operational Research, 212*(2), 277–286.

Rosenberg, O., & Ziegler, A. (1992). A comparison of heuristic algorithms for cost-oriented assembly line balancing. *ZOR – Methods and Models of Operations Research, 36*(6), 477–495.

Rubinovitz, J., & Levitin, G. (1995). Genetic algorithm for assembly line balancing. *International Journal of Production Economics, 41*(1–3), 343–354.

Salveson, M. E. (1955). The assembly line balancing problem. *Journal of Industrial Engineering, 6*(3), 18–25.

Scholl, A. (1993). Data of assembly line balancing problems. *Schriften zur Quantitativen Betriebswirtschaftslehre, 16*, TH Darmstadt.

Scholl, A. (1999). *Balancing and sequencing assembly lines*. (2nd ed.). Heidelberg: Physica.

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*(3), 666−693.

Scholl, A., & Klein, R. (1997). SALOME: A bidirectional branch and bound procedure for assembly line balancing. *INFORMS Journal on Computing, 9*(4), 319−334.

Scholl, A., Boysen, N., & Fliedner, M. (2008). The sequence-dependent assembly line balancing problem. *Operations Research Spectrum, 30*(3), 579−609.

Scholl, A., & Boysen, N. (2009). Designing parallel assembly lines with split workplaces: Model and optimization procedure. *International Journal of Production Economics, 119*(1), 90−100.

Scholl, A., Boysen, N., & Fliedner, M. (2009). Optimally solving the alternative subgraphs assembly line balancing problem. *Annals of Operations Research, 172*(1), 243−258.

Scholl, A., Fliedner, M., & Boysen, N. (2010). Absalom: Balancing assembly lines with assignment restrictions. *European Journal of Operational Research, 200*(3), 688−701.

Schwindt, C. (1998). Generation of resource-constrained project scheduling problems subject to temporal constraints. *Report WIOR – 543*. Karlsruhe University.

Talbot, F. B., Patterson, J. H., & Gehrlein, W. V. (1986). A comparative evaluation of heuristic line balancing techniques. *Management Science, 32*(4), 430–454.

Thesen, A. (1977). Measures of the restrictiveness of project networks. *Networks, 7*(3), 193–208.

Urban, T., & Chiang, W. (2006). An optimal piecewise-linear program for the U-line balancing problem with stochastic task times. *European Journal of Operational Research, 168*(3), 771–782.

Uyar, Ş., & Uyar, H. T. (2009). A critical look at dynamic multi-dimensional knapsack problem generation. In M. Giacobini et al. (Eds.), *Applications of evolutionary computing, 5484/2009* (pp. 762–767). Berlin Heidelberg: Springer.

Wee, T. S., & Magazine, M. J. (1981). An efficient branch and bound algorithm for an assembly line balancing – Part I: Minimize the number of work stations. *Working paper, No. 150*. University of Waterloo, Waterloo.

APPENDIX A.1

Actually, each run of our random experiment could be seen as a draw of some solution from the solution space with replacement. The probability for occurrence of optimal solutions is unknown, but the same in each run (or draw, or trial). The trials are statistically independent from each other. Therefore, the number of optimal solutions found by our random search runs is distributed binomially (no matter, what is the underlying distribution of the optimal solutions in the solution space).

Although no efficient exact procedure is known to compute the binomial confidence intervals, there exist a number of good approximations for it. We chose the Wilson score interval, recommended by Brown et al. (2001):

$$\frac{\hat{p}+\frac{1}{2n}\cdot z_{1-\alpha/2}^2 \pm z_{1-\alpha/2}\cdot\sqrt{\frac{\hat{p}(1-\hat{p})}{n}+\frac{z_{1-\alpha/2}^2}{4n^2}}}{1+\frac{1}{n}\cdot z_{1-\alpha/2}^2}, \qquad \text{(A1)}$$

where $\hat{p}$ is the share of optimal solutions found, $z_{1-\alpha/2} = 1.96$ is for the 95% confidence interval and $n = 10{,}000$ runs were taken.

APPENDIX A.2

We derive the upper bound on the number of direct and indirect arcs $A_{max}(\mathbf{s})$, knowing that at given distribution of tasks to stages s_1, ... s_K the maximal number of direct and indirect arcs is as follows:

$$A_{max}(\mathbf{s}) = 2\cdot\frac{\sum_{k=2}^{K} s_k\cdot\sum_{j=1}^{k-1} s_j}{2} = \frac{\sum_{k=2}^{K} s_k\cdot\sum_{j=1}^{k-1} s_j+\sum_{k=1}^{K-1} s_k\cdot\sum_{j=k+1}^{K} s_j}{2} = \frac{n^2-\sum_{k=1}^{K} {s_k}^2}{2} \qquad \text{(A2)}$$

From $OS_{max} = \frac{2\cdot A_{max}(\mathbf{s})}{n\cdot(n-1)}$, we get $OS_{max} = \frac{n^2-\sum_{k=1}^{K} {s_k}^2}{n\cdot(n-1)}$.

Obviously, OS_{max} only depends on the different values of s_k, irrespective to which stage these numbers are assigned. So, the same maximal order strength might be achieved for very different stages forms.

APPENDIX A.3: Proof for Theorem 1

Proof for OS_{min}: As argued above, each task on a stage k has at least $k-1$ (direct or indirect) predecessors. Since, according to the definition, each stage must contain at least one task, then the lower bound on the number of direct and indirect arcs is $\frac{K\cdot(K-1)}{2}$. This lower bound is sharp, since it is achieved if $n-K+1$ tasks are assigned to the first stage and only one task to any other of the remaining $K-1$ stages. If all but one tasks of the first stage have no successors, then there is only a single chain with $\frac{K\cdot(K-1)}{2}$ direct and indirect arcs connecting the K stages. Setting this value in proportion to the number of arcs in a complete precedence graph $\frac{n\cdot(n-1)}{2}$ results in $OS_{min} = \frac{K\cdot(K-1)}{n\cdot(n-1)}$.

Proof for OS_{max}: Given the number of tasks per stage s_k, $k \in \{1,\dots,K\}$, the achievable order strength is restricted to $OS_{max}(\boldsymbol{s})$ as defined in Appendix A.2.

Let us first assume that the average number of tasks per stage $\frac{n}{K}$ is integral. To determine a valid upper bound on the order strength, we take into account the constraint on the total number of tasks $\sum_{k=1}^{K} s_k = n$ in a Lagrangian optimization. Using a multiplier λ, we get from (A2) the following Lagrangian function to be maximized:

$$L(\boldsymbol{s},\lambda) = \frac{n^2-\sum_{k=1}^{K} {s_k}^2}{2} + \lambda\cdot\left(\sum_{k=1}^{K} s_k - n\right),\ \sum_{k=1}^{K} s_k = n \qquad \text{(A3)}$$

Computing partial derivatives for all variables and setting to zero yields:

$$\frac{\partial L}{\partial s_k} = -s_k + \lambda = 0 \Rightarrow s_k = \lambda\ \forall k\ \Rightarrow \sum_{k=1}^{K} s_k = K\cdot\lambda = n\ \Rightarrow\ \lambda = \frac{n}{K} \Rightarrow s_k = \frac{n}{K}\ \forall k$$

Examining the second-order conditions confirms that $s_k = \frac{n}{K}\forall k$ indeed is a maximum. Hence, if $\frac{n}{K}$ is an integer number, we get the maximal order strength if each stage contains the same number $s_k = \frac{n}{K}$ of tasks: $OS_{max} = \frac{n^2}{K^2}\cdot\frac{K\cdot(K-1)}{n\cdot(n-1)} = \frac{n\cdot(K-1)}{K\cdot(n-1)}$.

If $\frac{n}{K} \notin \mathbb{N}$, the maximal order strength is only achievable, if $\theta = n - \left\lfloor\frac{n}{K}\right\rfloor\cdot K$ stages

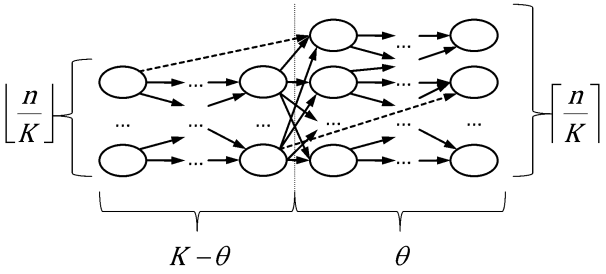


Figure A.1: *Illustration for the proof of maximal OS in the discrete case (examples of transitive arcs are dotted)*

each contain $\left\lceil \frac{n}{K} \right\rceil$ tasks and the other stages each include $\left\lfloor \frac{n}{K} \right\rfloor$ tasks (see the separate proof in Appendix A.4). Without loss of generality (see Appendix A.2), we assume the stages form as outlined in Figure A.1. Thus, we get a maximum of $\binom{\theta}{2} \cdot \left\lceil \frac{n}{K} \right\rceil$ (direct and indirect) arcs in the right block, $\binom{K-\theta}{2} \cdot \left\lfloor \frac{n}{K} \right\rfloor$ arcs in the left block and $(K-\theta) \cdot \left\lfloor \frac{n}{K} \right\rfloor \cdot \theta \cdot \left\lceil \frac{n}{K} \right\rceil$ arcs between the two blocks. So the order strength cannot exceed $OS_{max} = \frac{\left[\binom{\theta}{2} \cdot \left\lceil \frac{n}{K} \right\rceil + \binom{K-\theta}{2} \cdot \left\lfloor \frac{n}{K} \right\rfloor + (K-\theta) \cdot \left\lfloor \frac{n}{K} \right\rfloor \cdot \theta \cdot \left\lceil \frac{n}{K} \right\rceil\right]}{\binom{n}{2}}$.

Proof for maximal flexibility. For the sake of simplicity, we ignore the integrality requirement for the number of tasks at each stage and get $OS_{max} - OS_{min} = \frac{n \cdot (K-1)}{K \cdot (n-1)} - \frac{K \cdot (K-1)}{n \cdot (n-1)}$. Taking the derivative by K for this expression, we receive the first order condition for the maximum flexibility: $n^2 = K^2 \cdot (2K-1)$.

APPENDIX A.4: Proof for maximal order strength in case of non-integral $\frac{n}{K}$

In (A2), we have to correct the number of tasks of each stage for the optimal solution (in the continuous case) up by some $\varphi_k > 0$ or down by some $\varepsilon_k > 0$:

$$OS_{max} = \frac{n^2 - \sum_{k=1}^{K} \left(\frac{n}{K} - \varepsilon_k + \varphi_k\right)^2}{2}, \text{ with } \varepsilon_k \cdot \varphi_k = 0 \; \forall k \qquad \text{(A4)}$$

For each stage some correction must be present. Because we correct $\frac{n}{K}$ to an integer number at each stage k, either $\varepsilon_k = 0$ and $\varphi_k \geq \left\lceil \frac{n}{K} \right\rceil - \frac{n}{K}$ or $\varepsilon_k \geq \frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor$ and $\varphi_k = 0$.

Let $\varepsilon_k > 0$ for l stages and $\varphi_k > 0$ for $K - l$ stages. Then the lower bounds on the sums of corrections are $\sum_{k=1}^{K} \varepsilon_k \geq l \cdot \left(\frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor\right)$ and $\sum_{k=1}^{K} \varphi_k \geq (K-l) \cdot \left(\left\lceil \frac{n}{K} \right\rceil - \frac{n}{K}\right)$. Since $\sum_{k=1}^{K} s_k = n$, it follows $k=1K\varepsilon k=k=1K\varphi k$ and we search for a sharp lower bound. In the first case, $l \cdot \left(\frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor\right) \geq (K-l) \cdot \left(\left\lceil \frac{n}{K} \right\rceil - \frac{n}{K}\right)$, hence $l \geq \left\lceil \frac{n}{K} \right\rceil \cdot K - n$ and $\sum_{k=1}^{K} \varphi_k = \sum_{k=1}^{K} \varepsilon_k \geq \left(\left\lceil \frac{n}{K} \right\rceil \cdot K - n\right) \cdot \left(\frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor\right)$. In the second case, we receive the same lower bound on the amount of correction $\sum_{k=1}^{K} \varepsilon_k$.

To find the **maximal** achievable order strength, we take into account the constraints on the minimal amount of correction: $\sum_{k=1}^{K} \varepsilon_k \geq \left(\left\lceil \frac{n}{K} \right\rceil \cdot K - n\right) \cdot \left(\frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor\right)$ and $\sum_{k=1}^{K} \varepsilon_k = \sum_{k=1}^{K} \varphi_k$. The first order conditions we receive by taking l partial derivatives in ε_k and $(K-l)$ partial derivatives in φ_k from the following Lagrangian function:

$$L(\boldsymbol{\varphi}, \boldsymbol{\varepsilon}, \mu_1, \mu_2) = \frac{n^2 - \sum_{k=1}^{K} \left(\frac{n}{K} - \varepsilon_k + \varphi_k\right)^2}{2} + \mu_1 \cdot \left(\sum_{k=1}^{K} \varepsilon_k - \left(\left\lceil \frac{n}{K} \right\rceil \cdot K - n\right) \cdot \left(\frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor\right)\right) + \mu_2 \cdot \left(\sum_{k=1}^{K} \varepsilon_k - \sum_{k=1}^{K} \varphi_k\right), \quad \text{(A5)}$$

given $\sum_{k=1}^{K} \varepsilon_k = \sum_{k=1}^{K} \varphi_k$, $\mu_1 \cdot \left(\sum_{k=1}^{K} \varepsilon_k - \left(\left\lceil \frac{n}{K} \right\rceil \cdot K - n\right) \cdot \left(\frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor\right)\right) = 0, \mu_1 \geq 0.$ (A6)

By regrouping and taking into account $\varepsilon_k \cdot \varphi_k = 0$, we get:

$$L(\boldsymbol{\varphi}, \boldsymbol{\varepsilon}, \mu_1, \mu_2) = \frac{n^2}{2} - \sum_{k=1}^{K} \frac{n^2}{2 \cdot K^2} - \sum_{k:\, \varepsilon_k > 0} \frac{{\varepsilon_k}^2}{2} - \sum_{k:\, \varphi_k > 0} \frac{{\varphi_k}^2}{2} + \mu_1 \cdot \left(\sum_{k=1}^{K} \varepsilon_k - \left(\left\lceil \frac{n}{K} \right\rceil \cdot K - n\right) \cdot \left(\frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor\right)\right) + \mu_2 \cdot \left(\sum_{k=1}^{K} \varepsilon_k - \sum_{k=1}^{K} \varphi_k\right) \quad \text{(A7)}$$

From the first order conditions ($-\varepsilon_k + \mu_1 + \mu_2 = 0\,, \forall k | \varepsilon_k > 0$ and $-\varphi_k - \mu_2 = 0, \forall k \mid \varphi_k > 0$), we receive that:

If $\varepsilon_k > 0$ and $\varepsilon_{k'} > 0$ then $\varepsilon_k = \varepsilon_{k'}$ for all k and k', (A8)

If $\varphi_k > 0$ and $\varphi_{k'} > 0$, then $\varphi_k = \varphi_{k'}$ for all k and k', (A9)

$\mu_1 > 0$ and hence $\sum_{k=1}^{K} \varepsilon_k = \left(\left\lceil \frac{n}{K} \right\rceil \cdot K - n\right) \cdot \left(\frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor\right)$. (A10)

From our earlier considerations, that $l \geq \left\lceil \frac{n}{K} \right\rceil \cdot K - n$ and $\varepsilon_k \geq \frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor$ for these l stages, and (A10) it follows that $\varepsilon_k = \frac{n}{K} - \left\lfloor \frac{n}{K} \right\rfloor, \forall k | \varepsilon_k > 0$ and that there are $l = \left\lceil \frac{n}{K} \right\rceil \cdot K - n$ stages k with $\varepsilon_k > 0$.

The second order conditions ensure that we found the maximum.

In other words, we received an integer solution. The maximal achievable order strength could be received if $\left\lceil \frac{n}{K} \right\rceil \cdot K - n$ stages have $\left\lfloor \frac{n}{K} \right\rfloor$ tasks and the rest of the stages have $\left\lceil \frac{n}{K} \right\rceil$ tasks each.