FORMATO DE DISTRIBUCION MODULAR - ANÁLISIS Y BALANCEO

C.i. Nicole s.a.

ESTILO: ROWDY NUEVA 1-2012/2-2012	**PERS. BALANC.** 12	**CUOTA 100%**	851	**FECHA:**	14-dic-12
REFERENCIA: 607365-609204	**EFICIENCIA** 65 %		**NORMAL:**	**UNID HORA** 106	

COSTURA

OPERACIONES	MAQ	ESTANDAR	PERSONAS REQUERIDAS	STD
PREPARAR MARQUILLA	PRMAPL	0,1800	0.32	0,18
FIJAR MARQUILLA LATERAL	FIMLPL00	0,1970	0,35	0,197
MARQUILLAR TRASERO	MATRPL	0,2050	0.36	0,205
UNIR TIRI TRASERO	UNTTFI07	0,2870	0,51	0,287
UNIR TIRO DELANTERO	UNTDFI09	0,2510	0.44	0,251
CERR LADOS	CELAFI01	1,0710	1,90	1,071
AFINAR BOTAS	AFBOFI00	0,5730	1.02	0,573
REMATAR BOTA	REBOPL	0,352	0,62	0,352
DOBLADILLAR BOTAS ABIERTAS	DOBOAB	0,608	1.08	0,608
UNIR ENTREPIERNA	UNENFI03	0,6250	1,11	0,625
			0.00	0
RESORTAR CINTURA	RECIRE	0,2910	0,52	0,291
REMATAR ELASTICO	REELPL00	0,1260	0.22	0,126
DOBLADILLAR CINTURA	DOCIDO	0,4540	0,80	0,454
PULIR	PUXMN	0,5000	0,89	0,5
REVISAR	REPRMA	0,5000	0,89	0,5
DESMANCHAR Y REMALLAR	DEREMA	0,1010	0,18	0,101
EMPACAR	EMPRMA0	0,5510	0,98	0,551
TOTAL ESTILO		6,7710	12,00	6,7710

TABL.

6,8720 COST.

MAQ			
ASIGNACION		BONGO	
MINUTOS			
UND.CAPAC.	106		
MAQ		MAQ	
ASIGNACION		ASIGNACION	
MINUTOS		MINUTOS	
UND.CAPAC.	106	UND.CAPAC.	0
MAQ	PLA	MAQ	FIL4
ASIGNACION	12	ASIGNACION	11
REF		REF	
MINUTOS		MINUTOS	
UND.CAPAC.	106	UND.CAPAC.	106
MAQ	FIL4	MAQ	FIL4
ASIGNACION	9	ASIGNACION	10
CERRAR LADOS		CERRAR TIRO DELANTERO Y TRASERO	
MINUTOS	57	MINUTOS	57
UND.CAPAC.	53	UND.CAPAC.	106
MAQ	CLD	MAQ	FIL4
ASIGNACION	8	ASIGNACION	7
DOBLADILLAR BOTAS ABIERTAS		CERRAR LADOS	
MINUTOS	64	MINUTOS	57
UND.CAPAC.	106	UND.CAPAC.	53
MAQ	FIL4	MAQ	FIR
ASIGNACION	5	ASIGNACION	6
UNIR ENTREPIERNA		RESORTAR CINTURA	
MINUTOS	66	MINUTOS	31
UND.CAPAC.	106	UND.CAPAC.	106
MAQ	CLP	MAQ	PLA
ASIGNACION	4	ASIGNACION	3
REMATAR ELASTICO Y DOBLADILLAR CINTURA		REMATAR BOTAS MARQUILLAR LATERAL Y CINTURA	
MINUTOS	61	MINUTOS	21
UND.CAPAC.	106	UND.CAPAC.	106
MAQ	OFM	MAQ	OFM
ASIGNACION	1	ASIGNACION	3
REVISAR →		EMPACAR	
MINUTOS	106	MINUTOS	58
UND.CAPAC.	106	UND.CAPAC.	106

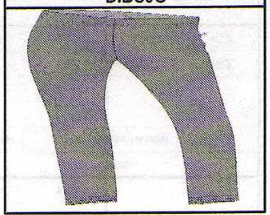
PREPARACIÓN

OPERACIONES	MAQ	ESTANDAR	PERSONAS REQUERIDAS	STD
		0,0000	0.00	0,0000
		0,0000	0.00	0,0000
		0,0000	0.00	0,0000
		0,0000	0.00	0,0000
		0,0000	0.00	0,0000
TOTAL PREPARACIÓN		0,0000	0,00	0,0000

MODULO

205

DIBUJO



METODOS

PUNTOS CRITICOS

ADITAMENTOS

C.i.Nicole S.A.S.

TOMA DE TIEMPOS TIPO

Código FING-01
Versión 1
No. Pag. 1 de 1
Fecha ~~28~~/04/2008

Fecha:
Modulo: 217
No. de Personas: 11
REFERENCIA legina
ESTANDAR COTIZADO 63%
META/HORA 143
LIDER lorena
SUPERVISORA Martina
HORA INIC. FINAL
CUMPLIMIENTO INICIAL 198%
CUMPLIMIENTO FINAL
PUNTOS DE MEJORA

CONVENCIONES	NOMBRE OPERARIA	OPERACION	SAM	MAQ	OBSERVACIONES 1	2	3	4	5	TIEMPO PROMEDIO	SUPLEMENTOS	VALORACION	TIEMPO TIPO	UNIDADES TIPO
1	Hector F.	Unir tiros D y T	0,469	FIL	~~0,57~~	0,45	0,46	0,46	0,45	0,455	19.3%	✓	0,542	110
2	Sandra	Unir tiros D y T	0,469	FIL	~~0,47~~	0,44	0,35	0,4	-	0,415	193%	✓	0,495	121
3	Andrea	Dobladillar ruedo		ML	0,41	0,4	0,35	—	—	0,386	188%	✓	0,459	131
4	Janetu	Dobladillar Ruedo		ML	0,35	0,3	0,3	-	-	0,35	188%	✓	0,415	145
5	Jairo	Unir entrepierna		FIL	0,5	0,5	0,5	-	-	0,5	193%	✓	0,596	100.
6	Alberto	Unir entrepierna		FIL	0,6	0,6	0,6	-	-	0,6	193%		0,715	84
7	Jorge	resortar pretina		FIR	0,7	0,7	0,7	-	-	0,7	193	-71	0,835	72 ✓
8	lorena	Marquillar		Pla	0,3	0,3	0,3	-	-	0,3	20%		0,36	167
9	Marieta	Asentar pretina		ML	0,65	0,65	0,65	-	-	0,65	188	-65	0,772	78 ✓
10	Gloria	Inspeccion		OFM	0,4	0,4	0,4	-	-	0,4	125%	-10	0,45	133 ✓
11	Andrea	Empacar		OFM	0,5	0,5	0,5	-		0,5	1205	-36	0,562	107 ✓
													4,589	

Sandra ✓ ok			Hector F. (+48)			Janet ✓ ok			Jairo			Alberto (+29)			lorena (+8)			Andrea		
Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq	
Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins
U. tiros	121	60	U. tiros	22	12	D. ruedo	145	60	U. ent	100	60	U. ent	43	31	Marquillar	143	52	Resoitar	71	60
			A. pret	62	48							Emp	36	20	A-pret	3	3			
Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq	
Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins
												Insp.	10	5						
Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq	
Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins
Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq		Operaria	Maq	
Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins	Operación	unds	mins

ESTUDIO DE TIEMPOS TIPO

TABLA RECONOCIMIENTO DE MAQUINARIA

MAQUINA	CONVENCION	Nº AGUJAS
PLANA	PLA	1 2
PLANA BARRA ESCUALIZABLE	PLE	2
PLANA PASA PUNTO	PPT	2
PLANA COSE CORTA	PCC	1
PLANA CADENETA	PLC	1 2
FILETEADORA	FIL	1 2
FILETEADORA RESORTADORA	FIR	1 2
PUNTADA INVISIBLE	PIN	1
COLLARIN	CLL	1 2 3
COLLARIN SESGADORA(2 & 3 Agujas)	CLS	1 2 3
COLLARIN DOBLADILLADORA	CLD	1 2 3
COLLARIN EXPUESTO	CLE	1 2 3
RIBETEADORA	RIB	1 2
PRESILLADORA	PRS	1
BROCHADORA AUTOMATICA	BRA	
BROCHADORA MANUAL	BRM	
AUTOMATA DE PEGAR BOLSILLO	AUB	1
PREHORMADORA DE BOLSILLO	PRB	
AUTOMATA DE DOBLADILLO	AUD	1 2
JUMPING	JUM	1
FUSIONADORA	FUS	
OJALADORA PLANA	OJP	1
OJALADORA LAGRIMA	OJL	1
BOTONADORA	BOT	1
FLAT SEAMER	FLS	3 4
CERRADORA DE CODO	CCO	2
ZIGZAG	ZIG	1
PICOETA	PIC	1 2
MULTIAGUJAS(ENTRE 4 y 12 AGUJAS)	MUL	1 a 12
OFICIO MANUAL	OFM	
PRENSA	PRE	
MESA DE VACIO	MES	
PREHORMADORA	PRH	
PARIS	PAR	
TOPPER	TOP	
LEGGER	LEG	

TABLA DE SUPLEMENTOS

MAQUINA	SUPLEMENTO
PLANA	20.0%
FILETEADORA	19.3%
COLLARIN	18.8%
COLLARIN SESGADORAS	18.8%
COLLARIN DOBLADILLADORA Y COLLARIN EXPUESTO	18.8%
FILETEADORA RESORTADORA	18.8%
PRESILLADORA	20.0%
FUSIONADORA	15.0%
OJALADORA	18.8%
BOTONADORA	18.8%
FLAT SEAMER	21.0%
CERRADORA DE CODO	18.8%
PLANA-ZIGZAG	20.0%
PICOETA	20.0%
MULTIAGUJAS(ENTRE 4 y 12 AGUJAS)	21.0%
OFICIO MANUAL	12.5%

FORMULA

$$\text{UNIDADES REALES HORA: } \frac{\text{MINUTOS HORA}}{\text{TIEMPO NORMAL * (1 + \% SUPLEMENTO)}}$$

TIEMPO NORMAL ; PROMEDIO CICLOS * % VALORACION OPERARIO.

ESCALA DE VALORACION

TOPOS DE ESFUERZOS	ITEMS	OBSERVACIONES
A) Esfuerzo deficiente 30%	. Pierde el tiempo claramente . Falta de interés en el trabajo. . Le molestan las sugerencias.	Dar vueltas innecesarias en busca de herramienta o material. Efectúa mas movimientos de los necesarios, mantiene en
B) Esfuerzo regular 50%	. Las mismas tendencias que el anterior pero en menor intensidad . Acepta sugestiones con poco agrado . Su atención parece desviarse del trabajo.	Es medianamente sistemático, pero no sigue siempre el mismo orden, trabaja también con demasiada exactitud, hace su trabajo demasiado dificil
C) Esfuerzo promedio 75%	. Trabaja con consistencia . Mejor que el regular . Es un poco escéptico sobre la honradez del observador de tiempos o de la dirección.	Tiene una buena distribución en su área de trabajo, planea de antemano, trabaja con buen sistema.
D) Esfuerzo bueno 100%	. Pone interés en el trabajo . Muy poco o ningún tiempo perdido . No se preocupa por el observador de tiempos	Está bien preparado y tiene en orden su lugar de trabajo
E) Esfuerzo excelente 125%	. Trabaja con rapidez . Utiliza la cabeza tanto como las manos . Toma gran interés en el trabajo	Reduce al mínimo los movimientos innecesarios, trabaja sistemáticamente con su mejor habilidad
F) Esfuerzo excesivo 150%	. Se lanza a un paso imposible de mantener constantemente . El mejor esfuerzo desde el punto de vista menos el de la salud.	

FORMATO DE DISTRIBUCIÓN MODULAR - ANÁLISIS Y BALANCEO

Montajes C.I. Nicole S . A

ESTILO: MALIBU BABY FONDO DUO 2-2012	PERS. BALANC. 12	CUOTA 100%	966	FECHA:	18-dic-12
REFERENCIA: 609182--609184	EFICIENCIA 65 %		NORMAL:	UNID HORA	121

COSTURA

OPERACIONES	MAO	ESTÁNDAR	PERSONAS REQUERIDAS
FIJAR MARQUILLA BANDERA	FIMABAPL	0,1420	0,29
FUSIONAR MARQUILLA TALLA	FUEMFU022	0,2610	0,53
UNIR HOMBROS	UNHOFI003	0,3410	0,69
RECOGER FRENTE	REFRPL14	0,2380	0,48
CERRAR CUELLO	CECUPL00035	0,2470	0,50
FIJAR CUELLO EN V	FICUPL006	0,2860	0,58
PEGAR CUELLO EN V	PECUFI00077	0,4590	0,92
PEGAR CINTA CUELLO	PECICC008	0,4100	0,83
SESGAR MANGAS	SEMACOSE	0,3570	0,72
MONTAR MANGAS	MOMAFI00010	0,4410	0,89
CERRAR LADOS MARQUILLANDO	CELAMA2	0,5350	1,06
DOBLADILLAR RUEDO	DORU2A0912	0,3960	0,80
PULIR PRENDA	PUPRMA13	0.5	1,01
REVISAR PRENDA	REPRMA14	0.5	1,01
DESMANCHE Y REMALLE	DEREMA15	0,0510	0,10
FIJAR MARQUILLA CODIGO DE BARR	FIMLPL31	0,1800	0,36
RECOGER COPA MANGAS	RECOPL	0,3530	0,71
ASENT CINTA DOBLA/ 2 EXT	ASCIPL	0,3170	0,64
			0,00
			0,00
			0,00
			0,00
			0,00
			0,00
			0,00
TOTAL ESTILO TABLERO		5,9630	12,0000
TOTAL ESTILO SEGUIMIENTO		6,0140	

TERMINACION

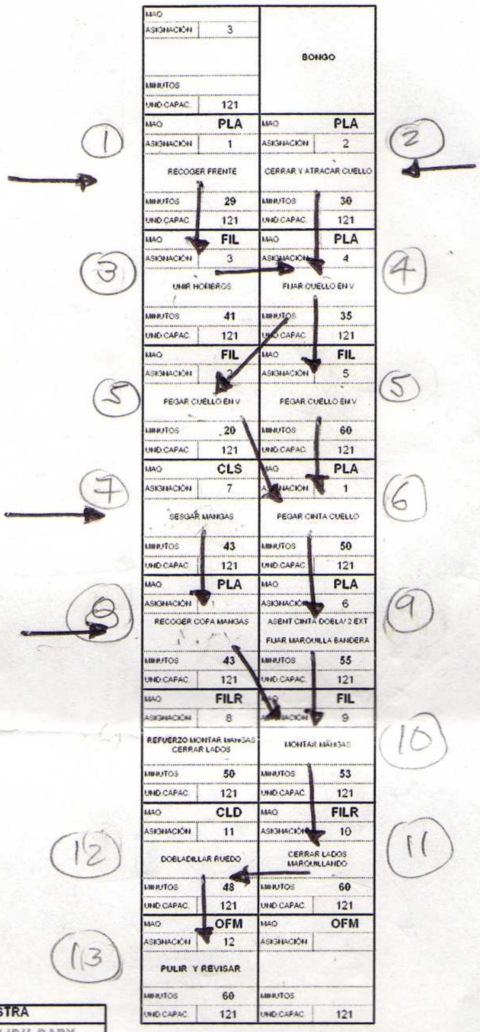
OPERACIONES	MAO	ESTANDAR	PERSONAS REQUERIDAS
LAVAR PRENDA	LAPRMA0	0.551	1,16
PULIR PRENDA	PUPRMA28	0.5	1,05
REVISAR PRENDA	REPRMA27	0.5	1,05
EMPACAR PRENDA	EMPRMA17	0.752	1,58
DESMANCHAR Y REMALLAR	DEREMA19	0.051	0,11
			0,00
			0,00
			0,00
TOTAL TERMINACION		2,3030	4,94

TERMINACIÓN

OPERACIONES	MAO	ESTÁNDAR	PERSONAS REQUERIDAS
		0.0000	0,00
		0.0000	0,00
		0.0000	0,00
		0.0000	0,00
		0.000	0,00
		0.000	0,00
TOTAL TERMINACIÓN		0,0000	0,00

MÉTODOS

PUNTOS CRÍTICOS



MAO		BONGO	
ASIGNACIÓN	3		
MINUTOS			
UND.CAPAC.	121		
MAO	PLA	MAO	PLA
ASIGNACIÓN	1	ASIGNACIÓN	2
RECOGER FRENTE		CERRAR Y ATRACAR CUELLO	
MINUTOS	29	MINUTOS	30
UND.CAPAC.	121	UND.CAPAC.	121
MAO	FIL	MAO	PLA
ASIGNACIÓN	3	ASIGNACIÓN	4
UNIR HOMBROS		FIJAR CUELLO EN V	
MINUTOS	41	MINUTOS	35
UND.CAPAC.	121	UND.CAPAC.	121
MAO	FIL	MAO	FIL
ASIGNACIÓN	2	ASIGNACIÓN	5
PEGAR CUELLO EN V		PEGAR CUELLO EN V	
MINUTOS	20	MINUTOS	60
UND.CAPAC.	121	UND.CAPAC.	121
MAO	CLS	MAO	PLA
ASIGNACIÓN	7	ASIGNACIÓN	1
SESGAR MANGAS		PEGAR CINTA CUELLO	
MINUTOS	43	MINUTOS	50
UND.CAPAC.	121	UND.CAPAC.	121
MAO	PLA	MAO	PLA
ASIGNACIÓN	1	ASIGNACIÓN	6
RECOGER COPA MANGAS		ASENT CINTA DOBLA/ 2 EXT FIJAR MARQUILLA BANDERA	
MINUTOS	43	MINUTOS	55
UND.CAPAC.	121	UND.CAPAC.	121
MAO	FILR	MAO	FIL
ASIGNACIÓN	8	ASIGNACIÓN	9
REFUERZO MONTAR MANGAS CERRAR LADOS		MONTAR MANGAS	
MINUTOS	50	MINUTOS	53
UND.CAPAC.	121	UND.CAPAC.	121
MAO	CLD	MAO	FILR
ASIGNACIÓN	11	ASIGNACIÓN	10
DOBLADILLAR RUEDO		CERRAR LADOS MARQUILLANDO	
MINUTOS	48	MINUTOS	60
UND.CAPAC.	121	UND.CAPAC.	121
MAO	OFM	MAO	OFM
ASIGNACIÓN	12	ASIGNACIÓN	
PULIR Y REVISAR			
MINUTOS	60	MINUTOS	
UND.CAPAC.	121	UND.CAPAC.	121

MUESTRA

MODULO

201

ADITAMENTOS

ELSEVIER

European Journal of Operational Research 91 (1996) 367-385 -■

EUROPEAN
JOURNAL OF
OPERATIONAL
RESEARCH

Theory and Methodology

Maximizing the production rate in simple assembly line balancing - A branch and bound procedure

Robert Klein, Armin Scholl *

Institut für Betriebswirtschaftslehre, Technische Hochschule Darmstadt, Hochschulstraße 1, D-64 289 Darmstadt, Germany

Received August 1994; revised January 1995

Abstract

In this paper, a branch and bound procedure for the Simple Assembly Line Balancing Problem Type 2 (SALBP-2) is described. This NP-hard problem consists of assigning tasks to a given number of work stations of a paced assembly line so that the production rate is maximized. Besides, possible precedence constraints between the tasks have to be considered. Existing solution procedures for SALBP-2 are mainly based on repeatedly solving instances of the closely related SALBP-1, which is to minimize the number of stations for a given production rate. The proposed branch and bound procedure directly solves SALBP-2 by using a new enumeration technique, the Local Lower Bound Method, which is complemented by a number of bounding and dominance rules. Computational results indicate that the new procedure is very efficient.

Keywords: **Scheduling; Assembly line balancing; Branch and bound**

1. Introduction

We consider the Type 2 of the Simple Assembly Line Balancing Problem (SALBP-2) which arises in the mass production of a *single product.* The assembly of each product unit requires the execution of n *tasks* (indivisible elements of work) with fixed *operation times* t_j ($j = 1,\dots,n$). *Precedence constraints* partially specify the order in which the tasks have to be performed. They can be represented by an acyclic precedence graph which contains nodes for all tasks with operation times as node weights and arcs (i, j) if task i has to be completed before task ; can be started. A *paced assembly line* consists of m *(work) stations,* connected by a conveyor belt onto which product units are launched at a constant rate p. Due to the uniform movement of the belt, the *production rate* is equal to p. The constant time interval l/p between the arrival of two consecutive units in all stations is called *cycle time* c. Each station $k = l,\dots,m$ has repeatedly to perform a subset S_k of the tasks on consecutive units. Such a set S_k is called *station load* of station k, and the sum of operation times of the contained tasks is referred to as *station time* $t(S_k)$. Since all tasks have to be performed completely, the cycle time must not be smaller than the maximum of all station times.

Using these assumptions, SALBP-2 is to find a partition of the set of all tasks into disjoint sta-

---

* **Corresponding author.**

0377-2217/96/515.00 ***SSDI***
0377-2217(95)00047-X

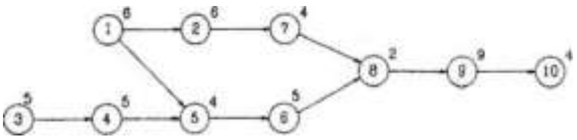


Fig. 1. Precedence graph.

tion loads S_k with k = \,2,...,m. For each arc (i, of the precedence graph the relation $h < k$ must hold if i e S_h and j eS_t, The objective is to maximize the production rate, or equivalently, to minimize the cycle time which is determined by the maximal station time.

Remark: For ease of presentation we assume that tasks are numbered according to a topological ordering, i.e. $i < j$ for all arcs (/, /). As objective we consider the minimization of the cycle time.

SALBP-2 is usually present when changes in the production process of a product take place. For example, operation times may be reduced by using alternative processing techniques. In this case, the number of stations of the line may remain fixed.

We consider an example with 10 tasks and the precedence graph of Fig. 1 (node weights indicate operation times). An optimal solution with m = 5 stations is the partition 5_X = {3, 4}, S_2 = {1, 5}, S_3 = {2, 7}, 5_4 = {6, 8}, S_5 = {9, 10} with the cycle time c = 13, which is determined by $t(S_s)$.

The sequel of the paper is organized as follows. In Section 2, we describe bound arguments for SALBP-2 which may be exploited by solution procedures. Section 3 surveys existing solution approaches. Most of these procedures rely on the close relationship of SALBP-2 to SALBP-1 which consists of minimizing the number of work stations for a given cycle time. Section 4 introduces SALOME-2, a new branch and bound algorithm, which is extended to a flexible bidirectional approach in Section 5. The results of computational experiments comparing existing as well as new procedures are summarized in Section 6. Finally, Section 7 contains conclusions which can be drawn from this research.

2. Bounds for SALBP-2

Solving SALBP-2 includes two main tasks which have to be accomplished simultaneously. First, a minimal cycle time has to be determined. Second, an assignment of all tasks to the m work stations with loads observing the precedence constraints and station times not exceeding the minimal cycle time has to be found. Such an assignment is called *feasible* for the respective cycle

Table 1
Definition of terms

number of tasks
operation time (task time) of task / — 1,..., *n* *Uj* is assumed to be positive and integral)
maximum task time; f_{min}: minimum task time sum of task times
set of tasks which immediately precede (follow, succeed) task in the precedence graph set
of all tasks which precede (follow, succeed) task *j* in the precedence graph number of
stations
station load, set of tasks assigned to station *k* = 1 , 2 , . . .
station time of station *k* (= £;- e s *tj)*
(realized) cycle time (= *max{t(S_k)* I *k* = 1,..., m))
lower (upper) bound on cycle time
idle time in station *k* (= c - *t(S_k))*
total available idle time for a realized cycle time *c (- m c —* f_{sum})
lower bound on the station time to realize cycle time c (= max{0, *c* — Totl(c)})
smallest integer >*x ; [x* J: largest integer
== \\(*tj* + *T.hf=P-th)/c*\\. earliest station of task *j* for cycle time c
•= *m* + 1 - \*Uj* + *e f th)/c*\\. latest station of task for cycle time c
= \*Ej(c)*, Ly(c)]: station interval of task ; for cycle time c

time. Hence, the problem can be solved by iteratively checking for several *trial cycle times* whether or not a feasible assignment of all tasks to *m* stations exists (cf. Section 3.1 for such solution procedures). This approach requires computing lower and upper bounds on the cycle time. Furthermore, it is possible to reduce the number of station loads which have to be considered for a certain trial cycle time by exploiting the problem structure of SALBP-2. The used terms are defined in Table 1.

Lower bounds for SALBP-2 can be obtained by utilizing relationships to other combinatorial optimization problems. By omitting the precedence constraints SALBP-2 passes into the problem of scheduling jobs (= tasks) on identical parallel machines (= stations) with the objective of minimizing the makespan (= cycle time). A simple *lower bound* LB1 for this problem (and for SALBP-2 as well) is obtained by allowing job *preemption,* i.e., processing of a job may be interrupted and continued on another machine. Since jobs cannot be processed on two machines simultaneously, the bound is given by

LB1 « max{$f_{\max}$, \ $t_{wam}/m]$ }.

For our example,

LB1 = max{9, **[50/51) =** ^{10}**-**

Another lower bound for the parallel machine problem related to SALBP-2 is obtained as follows. For ease of presentation, we assume that the tasks are numbered according to decreasing operation times, i.e.,

$tj > t_{)+l}$ for ; = 1, . . . , n - 1.

Consider the *m* + 1 largest tasks 1,..., m + 1. A lower bound on the cycle time for this reduced problem is $t_m + t_{m+x}$, the sum of the two smallest task times, because at least one station contains two tasks. In general, a *lower bound* LB2 is obtained by

n - 1
LB2.= **max^ £ f*.**$_{m+1—(}$ **I * =** 1,. *m*
, 1**= 0**

For our example, we get LB2 = 5 + 5 = 10 because the six largest tasks are 9, 1, 2, 3, 4, and 6 with operation times 9, 6, 6, 5, 5, and 5.

A main characteristic of SALBP-2 is the existence of precedence constraints. Though they complicate the problem, they provide information for additional problem reduction. They restrict the possible assignment of each task to a station interval which is bounded by an *earliest* and a *latest* station, respectively (cf. Talbot and Patterson, 1984). Depending on a trial cycle time c, values for the earliest and latest stations can be derived by

(',■ + E **0**/c
\ *hep,' I* **(la)**

(earliest station for task / = !,...,«), and

Lj(c) ■■= *m* + 1 (lb)A

(latest station for task *j* = 1,..., *n).*

Formula (la) takes into consideration that task ; must not start before all preceding tasks have been finished. A lower bound on the number of stations required for task ; and the tasks of the predecessor set *P** is obtained by dividing the corresponding sum of task times by the trial cycle time c. Equivalently to (la), formula (lb) considers the task times of task ; and all its successors.

Table 2
Earliest and latest stations for *m* = 5 and c = 11

	1	2	3	4	5	6	7	8	9 10
f .	g	6	5	£	4	5	4	2	9 4~
Ey(ll)	1	2	1	1	2	3	2	4	5 5
¿,(11) 2	3	2	3	3	4	4	4	4	5

;

If the trial cycle time c is to be realized, each task $j = 1,...,n$ must be assigned to a station in its *station interval*

$$Sl_j(c) = [E_j(c), L_j(c) \backslash .$$

Tasks whose station intervals enclose a certain station k are *potentially assignable* to k.

If we assume $c = 11$ to be the trial cycle time, we get the values of Table 2 for our example of Section 1. Since the station interval $SI_9(11)$ is empty for task 9, i.e., it cannot be assigned to any station, no feasible solution can exist for the trial cycle time 11. Note that due to £,(c) <$£_y$(c - 1) and $Lj(c) > L_y(c - 1)$ smaller cycle times cannot be feasible, too. Hence, the lower bound can be increased to 12 for the example.

In general, a *lower bound* LB3 for SALBP-2 can be defined as

LB3

— min{c I $Ej(c)$ < L,(c) for all $j = 1,...,n$ }.

For the trial cycle time $c = 12$, the values of Table 3 result.

A simple *upper bound* on the cycle time is

UB := max{f_{max}, $2 - [t_{sum}/m \backslash)$

(cf. Coffman et al., 1978, as well as Hackman et al., 1989). If LB1 > f_{max}, UB can be improved to LB1 + f_{max} - 1 due to the following reflections (cf. Scholl, 1995). It is always possible to determine a preemptive solution with cycle time LB1 in which each task is either processed by one station completely or is split up between two consecutive stations k and $k + 1$ (cf. Mc-Naughton, 1959). This theoretical upper bound implies the idea for a simple heuristic. Station 1 * is filled up by successively assigning tasks in the order of the topological task numbering until the station time is equal to or just exceeds the cycle time LB1. The same procedure is repeated for the stations 2, 3,..., m. For our example, the heuristic determines a solution with a realized cycle time of 13.

In the following, we outline some possibilities to utilize the bound arguments for reducing the number of station loads, which have to be examined in order to find a feasible assignment for a certain trial cycle time.

Since only trial cycle times c > LB1 are considered, the total time m ■ c available for processing a product unit equals or exceeds the sum of operation times t_{sum}. Hence, a *total idle time*

Totl(c) :=m- c -f_{sum}

cannot be used for work, i.e., one or more stations are not completely utilized. In such stations k, an idle time

$$I_k(c) \sim c - t(S_k)$$

occurs. Each of these idle times $I_k(c)$ for all k — 1,..., m is bounded by Totl(c). Therefore, a lower bound on the station times of all stations is

SL(c) := max{0, c - Totl(c)}.

Station intervals with only one element can be used to reduce SALBP-2 instances by *prefixing* because each task with $£_y$(c) = L,(c) must be assigned (prefixed) to station $£_;$(c) in order to find a feasible solution with cycle time c. In our example, the tasks 8, 9, and 10 are prefixed to the stations 4 and 5, respectively, for the trial cycle time c = 12 (cf. Table 3).

Furthermore, only maximal station loads have to be considered. A station load is *maximal* with respect to a cycle time c if no not yet assigned task can be added to the respective station neither exceeding the cycle time nor violating the

Table 3
Earliest and latest stations for $m = 5$ and $c = 12$

	j									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
E/12)	▌	▌	**1**	**1**	**2**	**3**	**2**	**4**	**4 5~**	
L/12) 2	**3**	**3**	**3**	**4**	**4**	**4**	**4**	**4**	**5**	

precedence constraints. In the example, the station loads S_4 = {8, 9} and S_5 = {10} obtained by prefixing are maximal with respect to $c = 12$, because none of the tasks 5, 6, and 7, which are potentially assignable to station 4, can be added to the load S_4. Task 10 is the only potentially assignable one for station 5. Hence, idle times $/_4(12) = 1$ and $7_5(12) = 8$ occur in stations 4 and 5.

3. Existing exact solution approaches

In this section, we provide a short summary of solution procedures for SALBP-2 presented in the literature so far.

3.1. Approaches using SALBP-1 procedures

Most approaches are based on the close relationship of SALBP-2 to SALBP-1 (minimize *m* for a given c). Both problem types can be reduced to a common *feasibility problem,* named SALBP-F. This problem is to find a feasible task assignment to *m* stations for a given cycle time c or to ascertain that none exists. Therefore, SALBP-2 can be solved by successively considering instances of SALBP-F with *m* stations and various trial cycle times of an interval [LB, UB] (cf. Section 2). Solutions for SALBP-F are obtained by using modified procedures for SALBP-1.

Procedures for SALBP-1

Since most research in simple assembly line balancing focussed on SALBP-1, a large number of exact solution procedures have been proposed (cf. Baybars, 1986, Domschke et al., 1993, as well as Scholl, 1995, for surveys and comparisons). For heuristic procedures see Talbot et al. (1986) as well as Scholl and VoB (1994).

Most procedures exactly solving SALBP-1 are based on the branch and bound principle as well as on dynamic programming. In the last years, only branch and bound procedures, clearly outperforming the existing dynamic programming approaches, have been developed. Among these, the algorithms FABLE of Johnson (1988), EUREKA of Hoffmann (1992) and SALOME-1 of Scholl and Klein (1994) seem to be most effective. Further branch and bound procedures stem from Talbot and Patterson (1984), Saltzman and Baybars (1987) as well as Hackman et al. (1989). Some of the dynamic programming procedures are those of Jackson (1956), Held et al. (1963), and Schräge and Baker (1978).

FABLE enumerates solutions by successively assigning tasks to the stations 1, 2,... according to a priority list. The enumeration is organized as a depth-first-search. Each station is maximally loaded before a new one is opened. Various dominance and bounding criteria are used for reducing the size of the enumeration tree.

EUREKA starts with a simple lower bound SB = f f_{sum}/cl on the number of stations and iter-atively solves instances of SALBP-F. If a solution with SB stations is found, the algorithm stops with the optimal solution of SALBP-1. Otherwise, SB is increased by one and the procedure is started once more, i.e., a new enumeration tree is built. Enumeration is done by systematically generating station loads (instead of assigning single tasks) in form of a depth-first-search. In a first phase of the algorithm, stations are considered in order of increasing numbers *(forward direction),* a second phase builds station loads in *backward direction* by reversing the precedence graph. Each phase is executed for a prespecified time interval with phase 2 starting after phase 1 and using the actual lower bound SB. If both phases fail to find an optimal solution, the heuristic of Hoffmann (1963) is applied which may result in a suboptimal solution.

SALOME-1 integrates and improves the most promising components of FABLE and EUREKA. Furthermore, some additional bounding and dominance rules as well as a new bidirectional branching strategy are included. This approach, which clearly outperforms FABLE and EUREKA, is adapted to solving SALBP-2 in Sections 4 and 5 of this paper.

Each of the algorithms described above can easily be modified in order to solve SALBP-F with *m* and c fixed. This is done by starting the procedures with the lower bound *m* and fathoming nodes whenever the lower bound has to be increased.

Search methods for SALBP-2

With respect to the sequence in which trial cycle times of the interval [LB, UB] are considered the following general search procedures are distinguished (cf., e.g., Mansoor, 1964, Wee and Magazine, 1981, as well as Hackman et al., 1989).

- *Lower Bound Method:* Starting with a lower bound LB, the cycle time is successively increased by one until the respective SALBP-F instance is feasible.
- *Upper Bound Method:* First, an upper bound UB on the cycle time is determined by a heuristic procedure, or a theoretical value is computed (cf. Section 2.1). Starting with UB, the cycle time c is successively decreased by one until SALBP-F is infeasible for c - 1 or c is equal to a lower bound. In each iteration, the maximal station time of the found feasible solution is used as new value of c.
- *Binary Search:* The search interval [LB, UB] is successively subdivided into two sub-intervals by choosing the mean element

$$c = [(LB + UB)/2J.$$

If SALBP-F is feasible for c, the upper bound UB is set to the maximum station time in the corresponding solution. Otherwise, LB is set to c +1. The search stops with an optimum cycle time UB when UB = LB.

- *Fibonacci Binary Search:* The method contains two steps. First, a Fibonacci Search is performed following the basic idea of the Lower Bound Method, i.e., the trial cycle times are LB, LB + F(l), LB + F(l) + F(2),... using the Fibonacci numbers F(l) ■= 1, F(2) ~ 2, and

$$F(i) ==F(J- 1) + F(/-2) \text{ for } i > 3.$$

If a SALBP-F instance with a trial cycle time c is not feasible, LB is set to c + 1. When a feasible cycle time c is obtained, the Binary Search is applied to the remaining interval [LB, $c]$,

- *Binary Search with Prespecified Entry Point:* The method differs from standard Binary Search only in determining the first trial cycle time. Based on the observation that for many problem instances the cycle times close to LB are more likely to be optimal than larger values, the first trial cycle time is determined by the maximal value of c for which $m = ff_{sum}/c1$ holds. The Binary Search is applied to the remaining search interval. In our example, the first trial cycle time is c = 12.

3.2. Direct procedures for SALBP-2

In contrary to SALBP-1, only few exact methods directly solving SALBP-2 are available.

Charlton and Death (1969) describe a general branch and bound procedure which is also able to solve flow and job shop problems with minor changes and does not sufficiently utilize the particular structure of assembly line balancing problems.

A specialized branch and bound procedure for SALBP-2 has been developed by Scholl (1994). It contains a heuristic procedure with a tabu search strategy for determining good initial upper bounds. Branching is performed as a depth-first-search by assigning a single task to a station in each step. The choice of the task-station-combinations is controlled by priority rules. In contrast to most other procedures for assembly line balancing problems, stations are not considered in a fixed order. The algorithm contains different ways of computing lower bounds exploiting the structural properties of SALBP-2 described in Section 2. The most effective bounding method is based on minimal idle times in every station which can be determined by solving particular knapsack problems. Furthermore, the algorithm makes intensive use of dominance and reduction rules.

4. SALOME **for** SALBP-2

We present a new branch and bound approach which directly solves SALBP-2. It is called SA-LOME-2 (Simple Assembly Line Balancing Optimization Method for Type 2).

4.1. Analysis of search methods

In order to examine the performance of the search methods presented in Section 3.1, the Lower Bound Method is applied to our example (cf. Fig. 2). The SALBP-F instances are solved by an enumeration procedure similar to that of EUREKA which systematically builds maximal loads. We start with the trivial lower bound LB1 = 10. Since Totl(lO) = 0, only station loads without idle time are feasible for SALBP-F with $c = 10$. Hence, only one station load is possible in the first three stations, respectively. The hatched node is fathomed because no load without idle time is available for station 4. For $c = 11$ the total idle time is 5 and the station times must be in the interval [6,11]. In the first branch of the respective tree, no idle time occurs until the station load $S_3 = \{5, 6\}$ with idle time $7_3(11) = 2$ is built. Hence, the *remaining total idle time* for the reduced problem with stations 4 and 5 is reduced to 3. Now, only loads with station times not smaller than 8 avoid exceeding the total idle time. Since only the maximal load $5_4 = \{6, 8\}$ with station time 7 exists for station 4, the respective hatched node is fathomed. The other hatched nodes are fathomed accordingly. After examining an enlarged tree for c = 12 without success, a feasible (= optimal) solution is found in the first branch of the tree for c = 13.

The example reveals some disadvantages of SALBP-1 based search methods:

- They do not use results of computations performed for previously considered trial cycle times (SALBP-F instances). Hence, large portions of enumeration trees may repeatedly be constructed. In our example, four trees containing common parts (boldfaced in Fig. 2) have to be built by the Lower Bound Method.
- Furthermore, these methods rely on the assumption that all values of the search interval are potential cycle times. This may lead to unnecessary iterations if no combination of tasks with a station time equal to a trial cycle time exists. For example, imagine the trivial case of a problem with operation times which are multiples of 100. Then, only multiples of 100 are candidates for the optimum cycle time.
- The practical application of solution procedures is often restricted by limited computation time. In this case, search methods, which examine infeasible SALBP-F instances first, may not provide a feasible solution at all.

4.2. Outline of the algorithm

In order to avoid the disadvantages stated above, a procedure directly solving SALBP-2 is developed. It contains a new enumeration tech-

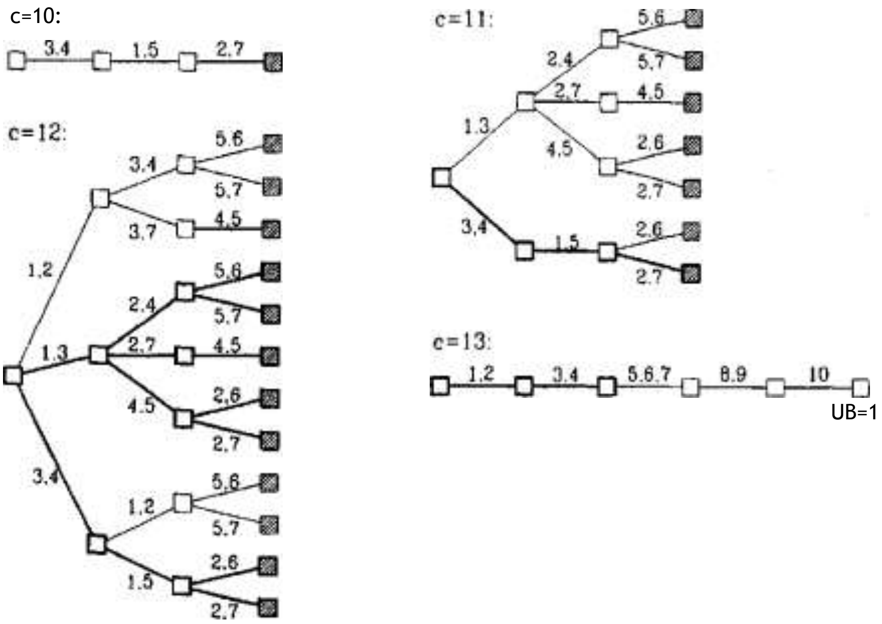


Fig. 2. Lower bound for the example problem.

nique, the *Local Lower Bound Method*, which exploits the observation that the relative difference between the minimal cycle time and an initial lower bound is very small for a large number of problem instances.

Branching is performed as a depth-first-search by successively building loads for the stations 1,..., *m*. Resulting subproblems (nodes of the enumeration tree) are reduced problems of the same type with less stations and a reduced precedence graph. The method starts with a lower bound LB on the cycle time and proceeds like a Lower Bound Method as long as possible in order to find a feasible solution with cycle time LB. Hence, in each node (which represents a partial solution of already built station loads) only such maximal station loads, whose idle times do not exceed the remaining total idle time (with respect to LB), are branched. After examining all loads feasible for the trial cycle time LB without success (no feasible completion of the partial solution with cycle time LB exists), the lower bound of the node is *locally* increased. Note that the Lower Bound Method fathoms this node. In order to avoid trying cycle times impossible for the reduced problem, the new value LB' of the *local lower bound* is determined as the smallest one for which at least one not yet considered station load is feasible. For LB' only those maximal loads which have not been feasible for former bound values are examined. Whenever loads for all stations but station *m* have been constructed, a feasible solution is obtained by assigning the remaining tasks to station *m*. The cycle time of the best known feasible solution serves as upper bound UB. A node is fathomed when its local lower bound exceeds the value of UB.

The outlined enumeration procedure is illustrated by means of our example. Fig. 3 shows the resulting enumeration tree, the nodes are numbered in order of their generation. The current local lower bounds are given as node weights

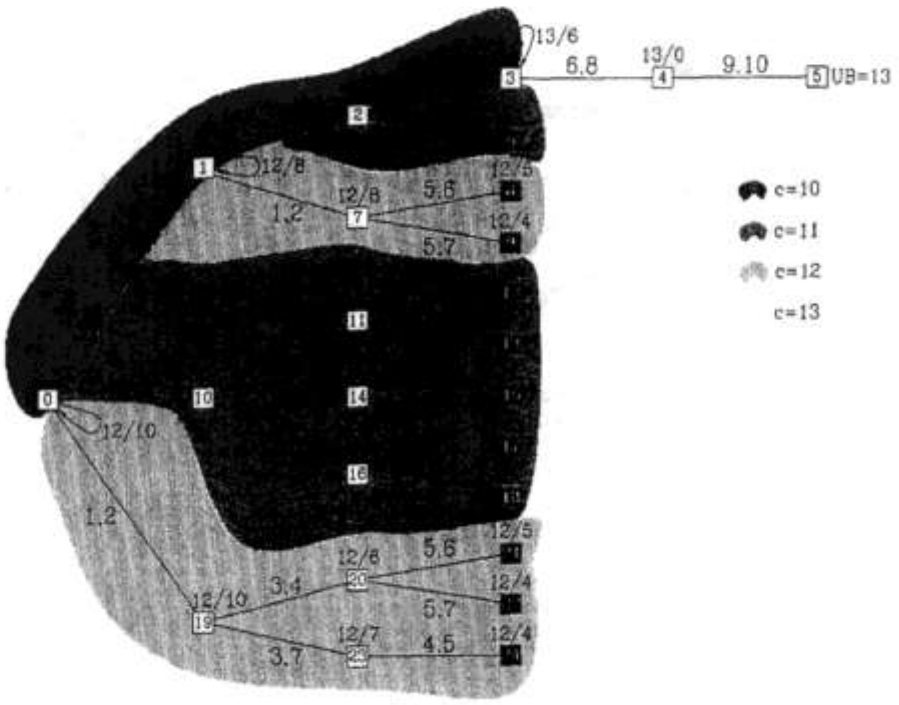


Fig. 3. Local lower bound method for the example problem.

together with the respective remaining total idle times. Loops mark the increase of a bound. Parts of the tree which correspond to the same value of the local lower bound are underlayed with the same pattern. In our explanation below, we denote the local lower bound of a node at *level k* of the tree, which represents a partial solution with stations *l , . . . , k ,* by LB^.

The procedure starts with the lower bound LB_0 = 10 as well as the total idle time Totl(lO) = 0 and builds the same station loads as the Lower Bound Method for SALBP-F with c = 10 (cf. Fig. 2). In node 3, the only maximal load 5_4 = {6, 8} for station 4 has an idle time $7_4(10)$ = 3. Hence, the partial solution S_l = {3, 4}, S_2 = {1, 5}, S_3 = {2, 7} cannot be completed to a feasible solution with cycle time 10. In contrast to solving a SALBP-F instance, the Local Lower Bound Method does not fathom the node but increases its local lower bound LB_3 to 13. The cycle times 11 and 12 are not possible in this node. For the cycle time 11 the first three station loads of the partial solution show idle times /jOD = $/_2(11)$ = $7_3(11)$ — 1. Since the only maximal load S_4 = {6, 8} of station 4 would result in the idle time $/_4(11)$ = 4, the cumulated idle time of 7 would exceed the total idle time TotI(ll) = 5. Due to idle times /j(12) = $/_2(12)$ - $7_3(12)$ = 2 and $7_4(12)$ = 5, the cycle time 12 with TotI(12) = 10 cannot be realized, too. With the local lower bound LB_3 = 13 a feasible solution is found in node 5 (UB = 13). Since the nodes 4 and 3 have the local lower bound value 13, the enumeration traces back to node 2 with the current local lower bound LB_2 = 10. This local bound must be increased because no further maximal load of station 3 exists for cycle time 10. Due to the station load S_3 = {2, 6}, the next trial cycle time is LB_2 = 11 with remaining total idle time 3. The resulting node 6 is fathomed after increasing its local bound to 13. The same is true for all other black nodes.

The example shows that the Local Lower Bound Method avoids repeatedly enumerating same parts of the tree, the main disadvantage of the Lower Bound Method (cf. Fig. 2). Furthermore, it provides a feasible solution in the first branch of the tree. Note that the new enumeration technique builds almost the same tree as the Lower Bound Method for *c* = 12 but generates station loads in another order according to increasing bounds.

```
Procedure Branch(k) begin
   if k = m - 1 then
   begin
      UB := LB*;
      fix all not yet assigned tasks to station m and store
      solution as current incumbent;
   end
   else
   while LB* < UB do       (* outer loop *) begin
      SL_k(LB_fc) := max{0, LB_k - TotI(LB*)
                                          k
                                      + 2>(LB»)};
                                        h = \
      find first maximal load Sk+i for station k + l   with
      r(5_t+i) € [SLjt(LBjt), LB*]; while Sk+i exists do
      (* inner loop *) begin
         LBjt+i := LBt;
         Branch(k + 1);
            (* recursive call of procedure branch*)
         find next maximal load Sk+i for station
         k + 1 with t(S_k+i) G [SLt(LBfc), LB*]; end
   increase LBt; end;
end;
```

Fig. 4. Procedure Branch.

In the following sections, we give a more detailed description of the enumeration technique of SALOME-2, the way of adjusting the local lower bounds and the used dominance rules.

4.3. The enumeration procedure

The algorithm starts with the global lower bound

LB := max{LBl, LB2, LB3}

and the upper bound UB heuristically determined as described in Section 2.

The enumeration is realized by the procedure 'Branch' in Fig. 4 which represents a recursive version of a depth-first-search branch and bound algorithm. The local lower bound LB_0 of the root

$(k = 0)$ is initialized to the value of the global bound LB before calling Branch(O). The local lower bounds LB^*_{+1} of nodes at level k + 1 are initially set to the value of the bound LB* of the father node at level k, respectively. In each node at level k of the enumeration tree, all possible maximal station loads for station k + 1 are enumerated explicitly or implicitly (cf. the inner loop of procedure Branch). In order to hold the current local lower bound LB*, only station loads feasible for the trial cycle time LB* are chosen for branching. Equivalently, the station times of those loads must be in the interval [SL*(LB*), LB*]. The lower bound SL*(LB*) is determined by the difference of LB* and the remaining total idle time. The latter results from reducing the total idle time TotI(LB*) by the idle times of the already built stations $l,\dots,k$ (cf. Section 2). The corresponding maximal station loads are systematically constructed by considering tasks in increasing order of their numbers. Note that the precedence graph is assumed to be topologically ordered and that the task order inside a station is not relevant for optimization. The enumeration scheme, which is not formulated in procedure Branch, is identical to that of EUREKA. In order to find good feasible solutions in the first branches of the tree, the tasks are renumbered by the renumbering procedure of FABLE described below.

After examining all station loads feasible for the current local lower bound LB* it has to be increased to a value LB'* (see Section 4.4 for details). This leads to an enlarged interval [SL*(LB'*), LB'*] which makes additional station loads feasible. Since the new interval includes the former one [SL*(LB*), LB*] as a subinterval, only not yet tried maximal loads with station times in the intervals [SL*(LB'*), SL*(LB*)) or (LB*, LB'*] are to be considered for the new local lower bound value LB*. The interval enlargement is repeatedly done until LB* is no longer smaller than UB (cf. the outer loop of procedure Branch).

The recursion terminates when loads for m — 1 stations have been built (cf. the first condition in procedure Branch). In this case, all remaining tasks are assigned to station m, and a solution with cycle time LB* results.

The renumberii.g procedure of FABLE which preserves a topological ordering is as fonVws. Initially, all tasks are not marked. In each of n iterations i = 1,..., n, one not marked task with largest operation time and no or only marked predecessors gets the number i and is marked. Ties are broken with respect to decreasing numbers of immediate successors and increasing original task numbers. In our example, the renumbering procedure leads to the original task numbers.

4.4. Adjusting lower bounds

In the enumeration procedure of Section 4.3, the incrementing of local lower bounds plays an important role. Due to the assumed integrality of task times, an obvious increment is 1. In order to avoid checking impossible trial cycle times and to enlarge the local lower bound by a minimum value concurrently, station loads not being part of the current station time interval are considered. Let 5^+ denote the smallest station time of a load S^*_{+1} larger than the cycle time LB* and $S\sim$ denote the largest station time smaller than SL*(LB*) of a load $S^*{}_{+1}$ which is maximal with respect to LB*. Using these terms, the next possible value LB* of the local lower bound LB*, is determined by formula (2):

(2)

$$LB'_t \bullet= \begin{cases} S^+ \quad \text{if } S \quad 0, \\ \min\left\{ S^+, \quad \text{'sum} \sim \wedge A - I'(\wedge A) - \$ \quad ?n-k-l \right. \\ \qquad \text{if } S->0. \end{cases}$$

In the case of $S\sim = 0$, no maximal load with a smaller station time than SL*(LB*) exists. Then only loads with station times larger or equal S^+ are left for branching and LB* can be set to S^+. Otherwise, it has to be checked whether the realization of a load with time 5^+ or 5" in station k + 1 leads to a lower increment of the local lower bound. If $t(S_{k+i}) = S\sim$ would be realized, the new value of LB* follows from applying the bound LB1 to the reduced problem with stations k + 2,..., m.

In our example, the initial bound of the root node 0 is LB_0 = 10 (cf. Fig. 3). Hence, Totl(lO) = 0 and SL_0(10) = 10. Only the load S_t = {3, **4}** is feasible for station 1 because it has station time 10. After examining the corresponding subtree LB_0 has to be increased. Due to the loads {1} and {1, 3}, the values of $S\sim$ and S^+ are 6 and 11, respectively. The new local lower bound is

LB_0==min{ll, f(50-6**)/4l}** = 11

with Totl(ll) - 5 and SL_0(11) = 11 - 5 = 6. Now, the only maximal load 5, = {1, 3} is used for branching. At the next revisit of the root node, the local bound is increased to S^+ = 12, the station time of the load S_y = {1, 2}, because no load which is maximal for cycle time 11 and has a station time smaller than 6 exists (5~=0). After examining the subtree with the load {1, 2}, the lower bound is increased to 5^+= 16, and the root node is fathomed because of UB = 13. In node 3, the new value of the local lower bound results from 5""= 7 and S^+ = 16. It is computed by

LB_3==min{16, [(50-30-7)/l]} =13.

Note that the values S^+ and $S\sim$ for adjusting the bounds can be determined while enumerating the loads which are feasible for the current local lower bounds, respectively.

4.5. Logical tests and fathoming

The Local Lower Bound Method already contains a bounding mechanism. In each node, branching is performed according to monotonously increasing values of the local lower bound LB_k. Hence, a node is fathomed whenever the value of LB_k reaches or exceeds UB, the cycle time of the incumbent solution. In this section, we describe further rules (dominance and reduction rules) which help to reduce the size of the enumeration tree.

Definition 1. A task which is not yet assigned to any station is called *available* for station $k + 1$ in a node at level k of the enumeration tree if each of the preceding tasks is already assigned to one of the stations 1,..., k, or is already contained in the load S_{k+1}.

Definition 2. A task h *potentially dominates* a task if Ff cf_{ft}* and tj^t_h hold. In the case of $tj = t_h$ and $F*$ =$F_{/l}$*, the lower indexed task potentially dominates the other one.

Maximum load rule. As already mentioned, only maximal loads have to be considered. As a consequence, all loads which can be extended by an available task without exceeding the current trial cycle time ***YS_k*** are to be excluded.

Jackson dominance rule. A load S_{k+l} can be excluded if there is an available task h which potentially dominates a task /'eS_{t+1} and

$t(S_{k+l})-tj + t_h<LB_k$

holds.

This rule is based on the one given by Jackson (1956). It uses the fact that all successors of task j are successors of task h as well and cannot start before task h is performed. Hence, the sequence of j and h is without consequence for the successors of /. The condition $tj < t_h$ of Definition 2 guarantees that the station time will not decrease if h replaces in S_{fc+1} and the second condition of the rule secures that the current lower bound LB_t remains valid.

In our example, task 4 is potentially dominated by task 1, task 6 by task 2, and task 7 by the tasks 3, 4, 5, and 6. The Jackson dominance rule avoids building nodes 14 and 23 due to the fact that task 7 can be replaced by the dominating task 4 in the respective loads for station 2. Furthermore, the nodes 9, 13, 18, and 22 would be avoided by the rule.

Extended maximum load rule. Consider the current branch of the enumeration tree which leads to a node at level k. Assume A_h to be the smallest operation time of a task which has been available in station h = 1,..., k. Then, for station $k + 1$ only loads with station times smaller than

$mm\{t(S_h) +A_h\backslash h = l,...,k)$

have to be considered, and the respective node can be fathomed if the local lower bound reaches or exceeds this value.

This rule takes into consideration that the load of lome station h loses the property of being maximal if in station i + 1 a load is realized which causes the local lower bound (i.e., the cycle time) to be larger or equal $t(S_h) + A_h$.

On the one hand, the extended maximum load rule results in large reductions of the enumeration tree in most cases. On the other hand, it may prevent finding feasible solutions in the first branches of the tree. The latter problem can be illustrated by the following example. Imagine, that a load S_t for station 1 is feasible for the lower bound LB_0 of the root node. Furthermore, we assume that no solution with load S_t and cycle time LB_0 exists and that the load S¡ can be extended by an available task to a load with station time LB_0 + 1 (at the next revisit of the root). Then, the extended maximum load rule fathoms all nodes of the subtree following S_x whenever their local lower bounds are increased. Hence, the procedure acts as a Lower Bound Method in this subtree and no feasible solution is found there. In the case of large problem instances, such subtrees may be very large so that no feasible solution is found within a prespecified time limit.

In order to simultaneously use the reduction capabilities of the extended maximum load rule and to find feasible solutions soon, the application of the rule is controlled by the following strategy which has been found by parameter adjusting.

- The rule is not used for the first m^2 nodes of the tree.
- Furthermore, the rule is not applied to a node at level k if

$\ln(UB-LB_0)/(m-/t)$

with the parameter

$a = 1.5-\ln(LB_0)/m$,

holds. This condition includes two aspects. First, it is desirable to improve the upper bound UB if the difference between UB and the (global) lower bound LB_0 is large. Second, it is not expensive to complete a solution when only few stations remain (small value of $m - k$). The bound difference is transformed into a logarithmic value in order to make it comparable to the station difference.

Permutation rule. In any node at level k, a load S_{k+1} does not need to be considered if

max{;' e S_{k+} J < min{j e S_k]

holds.

As a consequence of the topological numbering of tasks, the condition of the rule guarantees that no task in S_{k+l} is a successor of a task in S_k. Therefore, the loads S_k and S_{k+1} of stations k and $k + 1$ can be exchanged without violating precedence constraints. For example, consider the loads $S_x = \{3, 4\}$ and $S_2 = \{1, 2\}$ which lead to node 7 in the tree of Fig. 3. Node 7 can be fathomed because the load permutation $S_1 = \{1, 2\}$ and $S_2 = \{3, 4\}$ is also examined (node 20).

Prefixing tasks. All tasks ; with

£,(UB - 1) = $L_;$(UB - l) - $k + l$

are prefixed to station $k + 1$ in a node at level k.

The prefixed tasks must be assigned to the respective station in order to find a feasible solution with a cycle time smaller than UB.

The rules described above are applied to every subproblem at a level k in the following way. First of all, prefixing of tasks is performed which may lead to a problem reduction. Now, all possible loads of station k are enumerated. For each trial load the maximum load rule, the permutation rule and the Jackson dominance rule are applied in the given order. The remaining loads result in new subproblems which are treated in the same way. Whenever the local lower bound of a node is increased the extended maximum load rule is applied according to the above strategy. The original problem is solved to optimality when all nodes are completely branched or fathomed.

5. A bidirectional approach

In Section 4, we describe a version of our branch and bound procedure SALOME-2

13/4 13/1
: —LM--------[a]------13 ------ f.-? -TU UB=13
------------------- 2.1

Fig. 5. Backward enumeration for the example problem.

(hereafter called UniFor) which works from station 1 to *m* in an *unidirectional, forward oriented* manner. For several types of optimization problems bidirectional algorithms are shown to be more successful than unidirectional ones (cf., e.g., Lawler, 1991). As indicated by Scholl and Klein (1994) as well as Scholl and VoB (1994), this is the case for assembly line balancing problems, too. Hence, we describe a modification of SA-LOME-2 which uses a flexible bidirectional branching strategy.

An easy adaptation can be made to use the algorithm of Section 4 in a backward manner. This is achieved by applying it to the reversed precedence graph which results from reversing the directions of all arcs (cf., e.g., Saltzman and Baybars, 1987). Tasks are considered in order of decreasing task numbers during enumeration. In the sequel, this version of SALOME-2 will be named UniBack.

The enumeration tree of Fig. 5 is built by UniBack for our example. Only one branch, which represents the optimal solution S_5 = {10}, 5_4 = {9, 8}, 5_3 = {7, 6, 5}, S_2 = (4, 3}, S_1 = {2, 1}, is constructed because the lower bound LB_0 of the root node can immediately be increased to 12.

The example shows that the planning direction may have a considerable influence on the solution effort of the algorithm. In order to take advantage of this fact, it is desirable to be able to use both planning directions simultaneously in form of bidirectional branching. Simple bidirectional versions of SALOME-2, BiFor and BiBack, are obtained by considering stations in the order 1, *m*, 2, *m* - 1,..., \ (*m* + D/21, or *m*, 1, *m* - 1, 2,..., l(m + l)/2J,respectively. These strategies swap between *forward steps* (choose the first not yet considered station) and *backward steps* (choose the last not yet considered station). In backward steps, the reversed precedence graph is considered.

A more flexible approach (BiFlex) uses a priority based strategy to decide on the branching direction. Within this strategy, a mean value *T(k)* of operation times per station serves as priority criterion. For its definition we introduce

c - = UB- 1,

the maximum cycle time of an improved solution and the set

A $_k$ *(c)* - {;|*eSI,(c)}

of tasks potentially assignable to station *k* for the maximum cycle time c:

£,eWV(L^)-£;(c-) + l)) T{k) -

\ *A* $_k$ *(c)* \

for *k* = \, . . . , *m*. (3)

In (3), the operation times of all tasks *j*, which are potentially assignable to a station *k*, are

Table 4
Earliest and latest stations in the root node

	j									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
tj	**6**	**6**	**5**	**5**	**4**	**5**	**4**	**2**	**9**	**4**
Ej(c)	**1**	**1**	**1**	**1**	**2**	**2**	**2**	**3**	**4**	**4**
L/c)	**3**	**4**	**3**	**3**	**4**	**4**	**4**	**4**	**5**	**5**

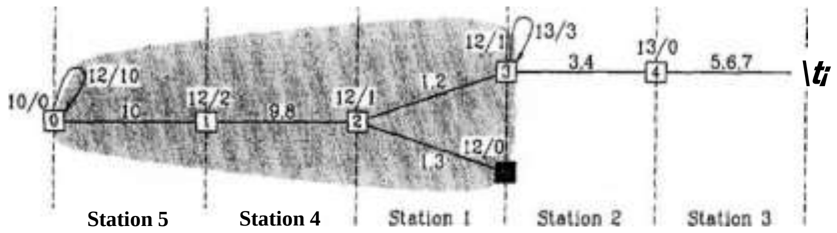


Fig. 6. Bidirectional enumeration tree for the example problem.

proportionately distributed among the stations of the station interval Sly(c). The sum of these parts is divided by the number of potentially assignable tasks resulting in a mean value $T(k)$ of the operation times of tasks for station k.

Let k_f and k_b be the next stations to be considered in forward and backward direction, respectively. Then, the following priority rule decides on the branching direction:

Perform a forward step, $if T(k_{\{}) > T(k_b)$ *or* $T(k_{\{}) = T(k_h)$ *and*

$\backslash A_{k<}(c)\backslash < \backslash A_{kb}(c)\backslash.$

In all other cases, make a backward step.

This simple rule is based on the longest processing time first rule which is successfully used in many unidirectional algorithms for SALBP-1 and other problems (cf. Scholl and Klein, 1994, for a foundation). It enables BiFlex to decide in each subproblem on the planning direction for the next step depending on the data of the respective reduced problem.

We illustrate the proceeding of BiFlex by our example and start with UB = 15 (c = 14). Table 4 contains the earliest and latest stations in the root node. Fig. 6 shows the resulting enumeration tree.

The first decision is to choose among the stations $k_f = 1$ and $k_h = m = 5$. Due to 7(l) = (f+ f+ f+ f)/4=1.71 and

7X5) = (f + |)/2 = 3.25,

a backward step is preferred. After increasing LB_0 from 10 to 12 and building the load S_5 = {10}, the values of L,(c) have to be modified (those of $£_y$(c) are not affected); see Table 5.

Because of 7(1) = 2.29 < 7(4) = 5, the station 4 is considered in a backward step. The load 5_4 = {9, 8} is branched and the latest stations remain valid for the not yet assigned tasks 1,..., 7. Hence, 7(1) keeps its value. Due to 7(3) = 2.03, the next step is a forward one. After assigning tasks 1 and 2 to station 1, the stations 2 and 3 as well as the tasks 3,..., 7 build the reduced problem with earliest stations $Ejic) = 2$ for / = 3, 4, 5, 7 and $E_6(c) = 3$ and the latest stations of Table 5. Because of 7(2) = 7(3) = 2.875 and $\mathrm{I} A_2(c) \mathrm{I} = \mathrm{I} A_3(c) |$ = 4, the tie breaker of the rule chooses station 2 in a forward step. Since no loads feasible for the local lower bound of 12 are available, it is increased to 13. For this cycle time the current partial solution can be completed by S_2 = {3, 4} and 5_3 = {5, 6, 7}. Only in node 2 another load (of station 1) feasible for cycle time 12 is available.

Table 5
Latest stations in node 1

;	i	2	3	4	5	6	7	8	9
Lj(c)	2	3	2	3	3	3	3	4	4

6. Computational results and

In this section, we report on results of computational experiments comparing existing algorithms and several versions of SALOME-2.

6.1. Data sets and experimental conditions

In the literature, no benchmark data sets are available for SALBP-2. We collected two data sets with 128 and 174 instances, respectively, which are based on *problems* (precedence graphs) described in the SALBP-1 literature. The data sets contain 9 and 8 different problems with 25 to 297 tasks. For each problem a range of station numbers is used to construct several *instances* (defined by a precedence graph and a number *m* of stations). The complete problem definitions and solutions are given in Scholl (1993).

All tests are performed on an IBM-compatible personal computer with 80486 DX2-66 central processing unit. Every tested algorithm is coded by means of Borland's Pascal 7.0 and is applied to each of the 302 problem instances of both data sets with a time limit of 500 CPU seconds. If the time limit is reached *{time out* for short) without proving the current incumbent solution to be optimal, only a heuristic solution is provided. Then, the obtained solution is characterized by the current global lower bound LB and the current upper bound UB. The quality of this solution can be measured by the relative deviations

UB
devUB := 100%

LB
devLB — **100%**

of the upper and the lower bound from the optimum cycle time *c* *, respectively. If c * is presently not known, the best available lower bound value is used for computing deviations.

The procedures are compared with respect to the following measures (based on 302 instances):

# opt. : Number of instances for which an optimal solution is found and proven.
# fail : Number of instances for which the initial feasible solution is not improved.
av.dev UB : Average relative deviation of UB from optimality in %. max.dev UB: Maximum relative deviation of UB from optimality in %. av.dev LB : Average relative deviation of LB from optimality in %. av.cpu : Average execution time in seconds.

6.2. Comparing several SALBP-1 based search methods

In Section 3.1, different general search methods which determine the minimal cycle time by solving SALBP-F instances for several trial cycle times are described. We report on a comparative evaluation of the Lower Bound Method (LBM), the Binary Search (BS), the Fibonacci and Binary Search (FBS) and the Binary Search with Pre-specified Entry Point (EBS).

As initial values for thé boundaries of the search interval [LB, UB], we take the lower bound

Table 6
Comparison of search methods

	LBM	BS	FBS	EBS
		126 15.95		
# opt.	150			
		99.96		
# fail	152 24.17			
		-0.28		
av.dev UB	99.96			
		264.24		
max.dev UB	-0.22			
av.dev LB	264.06	140		
av.cpu		32 7.10		
	141			
		99.86		
	0			
		-0.40		
	1.25			
		281.50		
	11.84			
	-0.40			
	282.50			
	150			

LB1 and the upper bound which is heuristically determined as described in Section 2 (Hackman et al., 1989, use a theoretical value of UB). Two different transformations are examined by considering the tasks in the original and the reverse ordering of their numbers, respectively. Since the initial UB is often far away from the optimal cycle time, we exclude the upper bound method from the test.

Within each of the four tested search methods, a version of FABLE which is modified to solve feasibility problems (SALBP-F) is applied. The results are summarized in Table 6.

Table 6 shows that LBM and FBS solve more instances to optimality than the other procedures because the lower bound cycle times LB1 are optimal for many problem instances. In these cases, LBM and FBS check only one trial cycle time while the other methods need more iterations. On the other hand, LBM fails to find a feasible solution at all in case of exceeding the time limit which occurs for more than half of the instances. Then, only the initial feasible solution with a worst case deviation of almost 100% is available. The same problem appears with FBS when it runs out of time in the Fibonacci Search phase. EBS fails to improve the initial solution if the first trial cycle time is too small or solving the respective SALBP-F instance takes too much time. Only BS finds feasible solutions for all instances. Consequently, it shows the best average and maximum deviations of UB from optimality. The worse values of the other methods result from the instances for which they are not able to improve the initial solutions. With respect to the average computation times (including 500 seconds in case of time out) LBM and FBS are better than the two other search methods because they find more optimal solutions and need fewer iterations for the respective instances. Furthermore, they result in better lower bound deviations.

The results can be summarized as follows. Only BS reaches acceptable solutions for all instances. While performing well for many instances, the search methods LBM and FBS fail to improve the initial feasible solution for harder problems. The choice of the particular first trial cycle time by EBS does not seem to be advantageous because this cycle time is not appropriate in some cases.

6.3. Comparing unidirectional and bidirectional versions of SALOME-2

In this section, we report on a comparison of the different versions UniFor, UniBack, BiFor, BiBack, and BiFlex of SALOME-2. The bounds LB and UB are determined as described in Section 6.2. In Table 7, we present a summary of the results.

The algorithm UniFor is outperformed by its mirror image UniBack. This indicates that the planning direction plays an important role with respect to the efficiency of the algorithm. The static bidirectional algorithms BiFor and BiBack are not able to improve the results of their unidirectional counterparts. On the contrary, they perform worse for some instances. The best results with respect to all criteria are obtained by BiFlex. Its flexible strategy is able to obtain solutions at least as good as the best of the four static algorithms for 260 of the 302 instances.

Table 7
Comparison of different planning directions

	UniFor	UniBack	BiFor	BiBack	BiFlex
# opt.	164	184	nï	*m*	2Î7
# fail	1	0	1	2	0
av.dev UB	1.09	0.90	1.24	1.51	0.56
max.devUB	10.05	11.73	53.22	53.22	10.03
av.dev LB	-0.33	-0.21	-0.24	-0.19	-0.12
av.cpu	242.74	208.71	228.99	210.92	157.03

6.4. Comparing SALOME-2 with existing procedures

We compare the flexible bidirectional version BiFlex of SALOME-2 (shown to be effective in Section 6.3) with four existing procedures. Three of them use modified SALBP-1 algorithms in the framework of Binary Search which appeared to be the best search method (cf. Section 6.2). The chosen SALBP-1 solvers are FABLE, EUREKA, and SALOME-1 (cf. Section 3.1). Furthermore, a version of the taskoriented branch and bound procedure of Scholl (1994) which directly solves SALBP-2 is included in the test (TBB for short, cf. Section 3.2). FABLE is applied as described in Section 6.2. Since EUREKA contains several parts, the remaining computation time is equally split up between the forward and the backward procedure in each iteration of the search process. If the time limit is reached only the heuristic procedure is applied for all remaining trial cycle times. This leads to total computation times exceeding the prespecified time limit of 500 seconds by about 7 seconds on average.

The algorithms SALOME-1 and SALOME-2 reflect the relationship between SALBP-1 and SALBP-2. While SALOME-2 applies a Local Lower Bound Method with respect to the cycle time, SALOME-1 uses a very similar technique for the number of stations. In both cases, SALBP-F instances arise for temporarily fixed local lower bounds on the cycle time and the number of stations, respectively. Because of this close relationship between the two methods, they contain common components like the procedure for enumerating station loads, most of the dominance and reduction rules and the flexible bidirectional branching strategy. The main differences are that SALOME-2 avoids repeated enumeration of parts of the tree and that it considers only possible cycle times. SALOME-1 is only used for solving feasibility problems within a Binary Search, i.e., no local bound enlargement takes place.

The algorithm TBB originally contains a heuristic procedure with a tabu search meta strategy to obtain initial feasible solutions. In order to have equal start conditions, the heuristic is not used, and all algorithms start with the same lower and upper bounds as in the previous sections. Scholl (1994) describes a number of priority rules for selecting the next task-station-combination to be fixed in a node of the enumeration tree. Among these, one similar to the bidirectional strategy of SALOME-2 is chosen because it clearly outperforms the other rules.

The five algorithms are applied to the combined data set with 302 instances in the same way as described in Section 6.2. The results are summarized in Table 8 (cf. Tables 6 and 7 for the results of FABLE (column BS) and SALOME-2 (column BiFlex), respectively).

The algorithms SALOME-1 (with Binary Search) and SALOME-2 show quite similar results because of their close relationship discussed above. Only with respect to the maximal deviation SALOME-2 has a significantly smaller value. Both methods clearly outperform the other procedures.

TBB does only well concerning the number of optimal solutions and the lower bound deviations. The large average and maximum deviations of UB from optimality are due to the enumeration scheme which orientates on improving the current UB instead of trying to hold a current lower bound like the SALOME procedures. Furthermore, it assigns only a single task in each branching step which requires the application of expen-

Table 8
Comparison of different algorithms

	FABLE	EUREKA	SALOME-1	TBB	SALOME-2
#opt. av.dev	**141**	**215**	**217**		
UB max.dev	**1.25**	**0.53**	**0.56**		
UB av.dev	**11.84**	**15.25**	**10.03**		
LB av.cpu	**-0.40**	**-0.11**	**-0.12**		
	282.50	**164.43**	**157.03**		
	178	**201**			
	0.62	**1.30**			
	11.86	**17.86**			
	-0.55	**-0.21**			
	293.34	**188.36**			

sive logical tests in order to avoid constructing badly utilized station loads.

The results of the SALBP-1 procedures FABLE, EUREKA, and SALOME-1 in the framework of Binary Search confirm the findings of Scholl and Klein (1994) who compared these algorithms for several data sets of SALBP-1 instances. SALOME-1 significantly outperforms the other procedures which is mainly due to its flexible bidirectional branching strategy and its sophisticated bounding and dominance rules. EUREKA's superiority to FABLE with respect to the number of optimal solutions and the upper bound deviations relies on the use of both planning directions as well as the heuristic. The better lower bound deviations and the shorter average computation times of FABLE can be explained by its bounding and dominance rules which are not included in EUREKA.

7. Summary and conclusions

In this paper, we present solution procedures for SALBP-2 which consists of balancing the work on an assembly line such that for a given number of stations the cycle time is maximized. For this problem type only few methods have been proposed in literature so far. Almost all approaches apply procedures for the strongly related SALBP-1, which is to minimize the number of stations for a given production rate, in the framework of a general search method. Effective solvers for this problem type are the well-known algorithms FABLE and EUREKA as well as a combination of these methods, called SALOME-1. The latter has been used to develop a procedure, named SA-LOME-2, which directly solves SALBP-2. Computational experiments show that this new approach clearly outperforms former methods. To our knowledge, this is the first time that SALBP-2 procedures are comprehensively investigated. Due to the lack of appropriate data sets, a new one with 302 instances has been constructed.

SALBP-2 as well as SALBP-1 are restricted versions of the general simple assembly line balancing problem (SALBP-G) which is to find a combination of the cycle time c and the number m of stations minimizing the sum of idle times. This problem has a nonlinear objective function depending on c and m. Two possible solution approaches consist of iteratively solving SALBP-1 instances for several values of c, or SALBP-2 instances for several values of m, respectively. Hence, further research may focus on examining whether SALBP-1 or SALBP-2 based procedures are more suitable for solving SALBP-G efficiently.

Acknowledgments

We wish to thank Professor Thomas Hoffmann and Professor Roger Johnson for providing us with copies of their Fortran codes of EUREKA and FABLE, respectively.

References

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

Charlton, J.M., and Death, C.C. (1969), "A general method for machine scheduling", *International Journal of Production Research* 7, 207-217.

Coffman, E.G., Garey, M.R., and Johnson, D.S. (1978), "An application of bin-packing to multiprocessor scheduling", *SIAM Journal on Computing* 7, 1-17.

Domschke, W., Scholl, A., and VoB, S. (1993), *Produktions-planung - Ablauforganisatorische Aspekte*, Springer-Verlag, Berlin.

Hackman, S.T., Magazine, M.J., and Wee, T.S. (1989), "Fast, effective algorithms for simple assembly line balancing problems", *Operations Research* 37, 916-924.

Held, M., Karp, R.M., and Shareshian, R. (1963), "Assembly line balancing - Dynamic programming with precedence constraints", *Operations Research* 11, 442-459.

Hoffmann, T.R. (1963), "Assembly line balancing with a precedence matrix", *Management Science* 9, 551-562.

Hoffmann, T.R. (1992), "EUREKA: A hybrid system for assembly line balancing", *Management Science* 38, 39-47.

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

Johnson, R.V. (1988), "Optimally balancing large assembly lines with 'FABLE' ", *Management Science* 34, 240-253.

Lawler, E.L. (1991), "Computing shortest paths in networks derived from recurrence relations", *Annals of Operations Research* 33, 363-377.

Mansoor, E.M. (1964), "Assembly line balancing - An improvement on the ranked positional weight technique", *Journal of Industrial Engineering* 15, 73-77; 322-323.

McNaughton, R. (1959), "Scheduling with deadlines and loss functions", *Management Science* 6, 1-12.

Saltzman, M.J., and Baybars, I. (1987), "A two-process implicit enumeration algorithm for the simple assembly line balancing problem", *European Journal of Operational Research* 32, 118-129.

Scholl, A. (1993), "Data of assembly line balancing problems", *Schriften zur Quantitativen Betriebswirtschaftslehre* 16/93, TH Darmstadt.

Scholl, A. (1994), "Ein B & B-Verfahren zur Abstimmung von Fließbändern bei gegebener Stationsanzahl", in: H. Dyckhoff, U. Derigs, M. Salomon and H.C. Tijms (eds.), *Operations Research Proceedings 1993*, Springer-Verlag, Berlin, 175-181.

Scholl, A. (1995), *Balancing and Sequencing of Assembly Lines*, Physica, Heidelberg.

Scholl, A., and Klein, R. (1994), "Combining the power of FABLE and EUREKA for assembly line balancing - A bidirectional branch and bound procedure", *Schriften zur Quantitativen Betriebswirtschaftslehre* 7/94, TH Darmstadt.

Scholl, A., and Voß, S. (1994), "Simple assembly line balancing - Heuristic approaches", *Schriften zur Quantitativen Betriebswirtschaftslehre* 2/94, TH Darmstadt.

Schräge, L., and Baker, K.R. (1978), "Dynamic programming solution of sequencing problems with precedence constraints", *Operations Research* 26, 444-449.

Talbot, F.B., and Patterson, J.H. (1984), "An integer programming algorithm with network cuts for solving the assembly line balancing problem", *Management Science* 30, 85-99.

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

Wee, T.S., and Magazine, M.J. (1981), "An efficient branch and bound algorithm for an assembly line balancing problem - Part II: Maximize the production rate", Working Paper No. 151, University of Waterloo, Waterloo, Ont.