IJIE
IBEROAMERICAN JOURNAL OF INDUSTRIAL ENGINEERING
REVISTA IBEROAMERICANA DE ENGENHARIA INDUSTRIAL
REVISTA IBEROAMERICANA DE INGENIARÍA INDUSTRIAL

APLICACIÓN DE LA METAHEURÍSTICA "SIMULATED ANNEALING" AL BALANCEO DE LÍNEAS DE ENSAMBLE SIMPLES

Leticia Capella[1]
Agustín Montagna[2]
Nélida Camussi[3]
Diego Cafaro[4]

RESUMEN: Una línea de producción y ensamble, en términos generales, es un agrupamiento de tareas en estaciones de trabajo que deben respetar determinadas relaciones de precedencia. El sistema productivo resultante es muy eficiente para la fabricación en masa de partes, componentes y ensambles. Uno de los desafíos de este sistema de producción es el denominado balanceo, que consiste en la asignación de las tareas a las estaciones de trabajo en función de la capacidad de producción y el tiempo productivo de la línea, de tal manera que la carga de las estaciones sea lo más equilibrada posible. Este trabajo presenta una experiencia positiva en la resolución del balanceo de líneas de ensamble simples (que manufacturan un solo producto) utilizando la metaheurística "simulated annealing". En la literatura se reporta muy frecuentemente la aplicación de otras metaheurísticas, como búsqueda tabú y algoritmos genéticos a este tipo de problemas, pero muy escasamente la aplicación de "simulated annealing", lo cual motivó el interés por implementarla. Esta metaheurística en particular posee características que la vuelven muy atractiva para su aplicación a casos de dimensiones reales.

Palabras claves: Simulated Annealing, Línea de Ensamble Simple, Balanceo de Líneas de Ensamble.

[1] Departamento de Ingeniería Industrial, Facultad de Ingeniería Química, Universidad Nacional del Litoral Santa Fe, Argentina. E-mail: leticapella@gmail.com
[2] Departamento de Ingeniería Industrial, Facultad de Ingeniería Química, Universidad Nacional del Litoral Santa Fe, Argentina. E-mail: agumontagna@gmail.com
3 Departamento de Ingeniería Industrial, Facultad de Ingeniería Química, Universidad Nacional del Litoral Santa Fe, Argentina. E-mail: ncamussi@intec.unl.edu.ar
4 Departamento de Ingeniería Industrial, Facultad de Ingeniería Química, Universidad Nacional del Litoral Santa Fe, Argentina. E-mail: dcafaro@fiq.unl.edu.ar

1 INTRODUCCIÓN

Las líneas de ensamble modernas constan de una serie de estaciones de trabajo conectadas entre sí por mecanismos manuales y/o automáticos de manejo de materiales, y presentan una alta eficiencia para la producción en masa. Uno de los desafíos de este sistema de producción es el de balancear la línea. Esto se aborda mediante la asignación de tareas (unidades indivisibles) a las estaciones de trabajo, de tal manera que será factible si se cumplen las relaciones de precedencia impuestas y si la suma de las duraciones de las tareas asignadas a cada estación no sobrepasa el tiempo de despacho de las piezas (tiempo de ciclo).

Un objetivo bastante común es la minimización del número de estaciones a tiempo de ciclo conocido. Si se logra un buen balanceo, las distintas estaciones quedan aproximadamente con una carga de trabajo similar. Se ha publicado una gran cantidad de trabajos para abordar este desafío aplicando procedimientos de solución tanto heurísticos como exactos (Scholl y Becker, 2006). Entre los heurísticos se menciona con bastante frecuencia la aplicación de búsqueda tabu a problemas de líneas de ensamble (Scholl y Voβ, 2006) como así también el uso de algoritmos genéticos (Rekiek y Delchambre, 2006). Dentro de los heurísticos, Dolgui y Proth (2010) mencionan a "simulated annealing"como una metaheurística adecuada para encarar este problema de optimización, el cual, entre otras características, posee múltiples soluciones óptimas alternativas. Manavizadeh y colab.(2013) lo aplicaron más recientemente a una línea de ensamble en forma de "U".

2 OBJETIVOS

Se pretende aplicar la metaheurística, llamada "simulated annealing", (a la que denotaremos como SA de aquí en más) propuesta por Kirkpatrick y colab. (1983) para resolver el problema de balanceo de líneas de ensamble simple. Se la programó usando la plataforma de desarrollo Matlab (2009) para el balanceo de líneas de ensamble de un solo producto, a fin de conocer el desempeño de la mencionada metaheurística a tales problemas. Las características primordiales de ese procedimiento son varias: (i) siempre trabaja sobre puntos factibles del problema que se está resolviendo, (ii) la estrategia de movimientos no es estrictamente descendente, sino que permite con una cierta "probabilidad" moverse a puntos

factibles eventualmente "peores" que la solución vigente a los fines de usar tales puntos como partida de nuevos caminos descendentes. Esa posibilidad es potencialmente muy favorable para problemas con múltiples óptimos locales.

3 APLICACIÓN DE SA A LÍNEAS DE ENSAMBLE DE UN SOLO PRODUCTO

3.1 Breve introducción a la metaheurística

Esta técnica está inspirada en el proceso de enfriamiento de los sólidos. Lo que busca realizar la estrategia es encontrar el mínimo de una función objetivo determinada.

Al aplicar este concepto al caso de líneas de ensamble, el algoritmo se concentra en generar una secuencia de soluciones, la cual no requiere necesariamente una mejora de la solución en cada paso dado. Por el contrario, puede aceptar una solución que es peor que la anterior con una probabilidad *p*. Esta probabilidad *p*, disminuye cuando el deterioro de la función objetivo crece y también a medida que van sucediendo las iteraciones. La meta de este enfoque es evitar quedar entrampado en el entorno de una solución local, pudiendo lograr, a partir de un nuevo punto, una solución eventualmente mejor.

El algoritmo tiene parámetros que marcarán sus movimientos iterativos sucesivos. Uno de los más significativos es la temperatura que descenderá gradualmente, por lo que hace falta definir un valor de temperatura inicial, cada cuántas iteraciones la temperatura disminuirá y un criterio con el cual se defina cómo se produce ese cambio de temperatura iteración a iteración.

A su vez, se debe definir una solución inicial factible a partir de la cual el algoritmo va a buscar moverse a otra solución en su entorno de manera tal que pueda minimizar una función objetivo determinada.

La elección de la temperatura inicial, está dada por un valor inicial T_0 con un valor “grande”, para que el sistema tenga cierto de libertad, y a su vez garantizar un número suficiente de iteraciones que permita alcanzar una “buena” solución (aunque nunca se puede hablar de optimicidad en metaheurísticas).

Se debe definir la evolución de la temperatura a medida que el algoritmo está iterando. La temperatura decrece cada K_n iteración, donde n representa el índice de la solución actual. Por lo tanto, K_n es el número máximo de iteraciones donde se aceptarán puntos después de

algún S_{n+1} "malo". Apenas detecta una mejora en la solución, K_n se reinicia y comienzan nuevamente las iteraciones. Algunos de los criterios para actualizar K_n son:

- $K_n = constante$
- $K_n = K_{n+1} + constante$, con K_0 dada.
- $K_n = {K_{n-1}}/{a}$, con $a < 1$
- $K_n = (K_{n-1})^{1/a}$ con $a < 1$
- $K_n = \frac{constante}{\ln(T_n)}$

Luego, se deben definir las reglas para la reducción de la temperatura. Se pueden considerar diferentes criterios para que la temperatura vaya disminuyendo, entre ellos se encuentran:

- $T_n = T_{n-1} - \text{constante}$
- $T_n = a\ T_{n-1}$, con $a < 1$
- $T_n = \text{constante}/(1 - n)$
- $T_n = \text{constante}/\ln(1 - n)$

Se debe proceder a la elección de la función objetivo, la misma puede ser el tiempo ocioso de la línea, la eficiencia de la línea, el número de estaciones de trabajo, etc. Y la misma va a estar denotada como $U(*)$, donde * es cualquier configuración factible de la línea.

Luego, se debe definir una solución inicial S_0. A partir de la cual el algoritmo va a comenzar a buscar mejores soluciones. Para esto, se puede aplicar algún procedimiento heurístico para obtener la solución S_0de partida y su respectivo valor de función objetivo $U(S_0)$. Cabe destacar que esta solución inicial así como las sucesivas, deben ser siempre factibles.

3.2 Concepto de “entorno” en líneas de ensamble

Para realizar esta búsqueda de una próxima configuración factible, el algoritmo se basa en la creación de otro “individuo” factible en el entorno de una solución factible. Para generar ese otro “individuo cercano pero diferente”S_{n+1} en el “entorno” de S_n, se proponen para el caso de líneas de ensamble las dos posibles operaciones:

- **Permutar dos operaciones ubicadas en dos estaciones adyacentes.** Para lo cual se elige aleatoriamente dos estaciones de trabajo contiguas y una tarea en cada una de ellas para realizar el intercambio. Una vez realizado el mismo, se debe verificar la factibilidad del cambio, es decir que no se supere el tiempo de ciclo ni se violen las relaciones de precedencia. Si la permutación no es posible, el algoritmo procede a elegir otras dos estaciones, y así hasta encontrar una solución factible. Se puede observar un ejemplo de permutación en la Fig. 1.

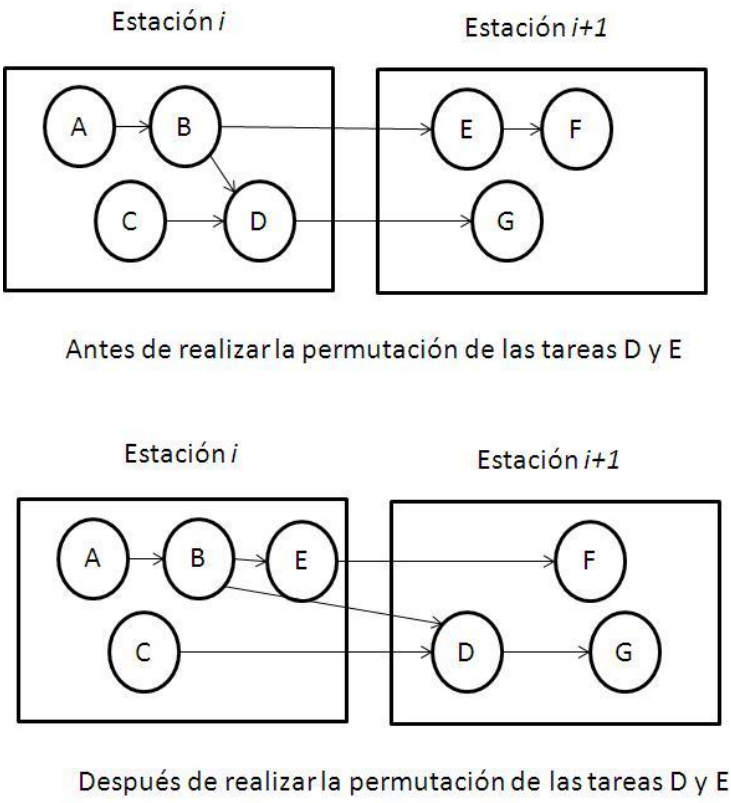

Fig.1. Ejemplo de una permutación entre dos estaciones de trabajo contiguas, en donde las relaciones de precedencia se respetan

- **Transferir una operación a la próxima estación.** Para ello, se procede a la elección aleatoria de una estación de trabajo y de una tarea dentro de la misma. Luego, se transfiere a la estación siguiente, y se verifica que la operación sea factible, es decir que no se incumpla la condición de superar el tiempo de ciclo ni

las relaciones de precedencia. Si la transferencia se aplica a una operación asignada a la última estación de la línea de ensamble, entonces se crea una nueva estación. Si una estación de la línea contiene solamente una operación antes de transferir, entonces esta estación desaparece, es decir se produce una fusión. Se puede observar un ejemplo de una transferencia en la Fig. 2.

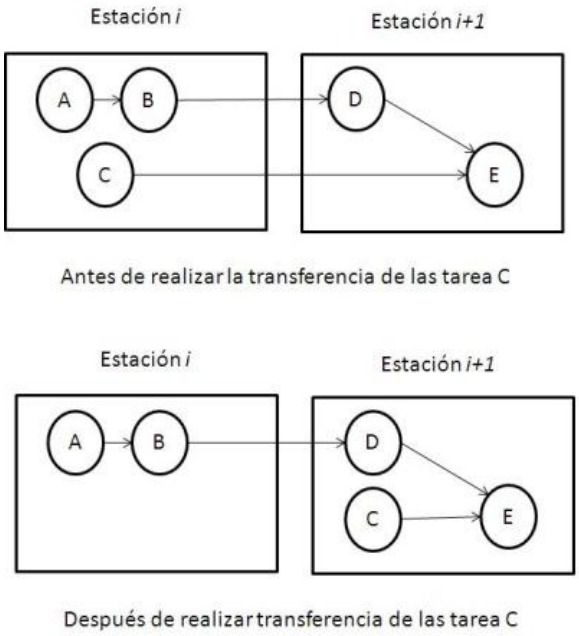

Fig.2. Ejemplo de una transferencia de una tarea entre dos estaciones contiguas, en donde se respetan las relaciones de precedencia.

Una vez encontrada una nueva configuración factible S_{n+1}y evaluado el correspondiente valor de función $U(S_{n+1})$, pueden suceder dos cosas: (a) $U(S_{n+1})$ es menor o igual a $U(S_n)$, es decir el valor objetivo de la nueva solución es mejor que la solución previa, entonces S_{n+1} es mejor solución que S_n (en un problema de minimización), por lo que se actualiza a S_{n+1} como la mejor solución disponible; (b)S_{n+1} no es mejor solución que S_n ($U(S_{n+1})$ es mayor a $U(S_n)$), la aceptación o rechazo de la nueva solución va a depender de la probabilidad *p*, la cual está dada por la Ecuación 1:

$$p_n = e^{-\Delta_n / T_n} \qquad (1)$$

donde, $\Delta_n = |U(S_{n+1}) - U(S_n)|$ y T_n es el parámetro decreciente llamada "temperatura". Luego para decidir si se acepta o se rechaza S_{n+1} cuando es peor que S_n, se debe generar un

número aleatorio x en el intervalo [0,1]; calcular p_n, y si p_n es mayor o igual que x, entonces se acepta a S_{n+1} como la próxima solución actual de la secuencia, caso contrario, se rechaza a S_{n+1} y se mantiene S_n como la solución vigente. Cabe destacar que la probabilidad p_n es una función decreciente a medida que avanza el algoritmo, por lo que en cada iteración va a ser menos probable aceptar una solución "mala".

Finalmente, se deben definir las reglas de terminación. Entre ellas se encuentran:

- Cuando la temperatura se hace menor que un valor dado ε
- Cuando el número de iteraciones excede un valor dado
- Cuando ninguna mejora ocurre después de una dada cantidad de iteraciones.

3.3 Pseudocódigo

En esta sección se provee el pseudocódigo utilizado para la resolución de problemas para líneas de ensamble

1. Introducir T, a, K, ε.

2. Generar una solución factible S_0 , calcular el correspondiente valor del criterio U(S_0) y adoptar S^*=S_0 , U(S^*) = U(S_0).

3. Sea k=0

4. k=k+1

5. Generar una solución factible aleatoria S_1 en el entorno de S_0 y calcular U(S_1).

6. Calcular: $\Delta = U(S_1) - U(S_0)$

7. Si $\Delta \leq 0$:

Poner $S_0 = S_1$ y U(S_0) = U(S_1).

Si U(S_1) < U(S^*) entonces $S^* = S_1$ y U(S^*) = U(S_1).

Si $\Delta > 0$:

Generar un valor aleatorio x ∈ [0,1] (distribución uniforme)

Calcular $p = \exp(-\Delta/T)$

Si x ≤ p, entonces $S_0 = S_1$ y U(S_0) = U(S_1).

8. Si k≥K, hacer:

T=aT

k=0

Si T ≥ ε, entonces volver a 4.

Si T <ε, finaliza el algoritmo y retorna la última solución obtenida.

3.4 Ejemplos de Aplicación

Para mostrar el comportamiento del algoritmo explicado, se presentan dos ejemplos de aplicación, el primero con 30 tareas y el segundo con 148 tareas.

Los dos casos se resuelven mediante SA programado en Matlab7.9.0 (R2009b), en un procesador Intel (R) Atom(TM) CPU @ 1.66 GHz 1.66 GHz, con una memoria instalada (RAM) de 0.98 GB, a partir de una configuración inicial S_0 dada.

Los parámetros que se tuvieron en cuenta para la resolución, en los dos ejemplos, fueron los siguientes:

- La temperatura inicial es $T_0 = 5000$.
- La temperatura decrece cada$K = 100$ iteraciones.
- El criterio para la disminución de la temperatura es de la forma:$T_n = a\, T_{n-1}$, donde la constante $a = 0.98$.
- El valor mínimo que se permite para la temperatura es $\varepsilon = 50$.
- Se considera que el número máximo de iteraciones que puede realizar el algoritmo es de 100000.
- Y, finalmente se considera que la cantidad permitida de iteraciones que ocurran sin que se produzca ninguna mejora es de 20000.

3.5 Ejemplo de aplicación con 30 tareas

Se considera un proceso de manufactura compuesto de 30 operaciones, denotadas por T1, T2,…, T30. En la Tabla 1 se presenta la lista de las tareas, sus relaciones de precedencia y la duración en minutos de cada una de ellas. Se considera en este problema untiempo de ciclo de 250 minutos.

Operación Elemental	Operaciones Precedentes	Tiempo (min)
T1	-	54

Operación Elemental	Operaciones Precedentes	Tiempo (min)
T2	-	144
T3	T1	138
T4	T1 , T2	144
T5	T3	24
T6	T3	102
T7	T4	150
T8	T7	114
T9	T5 , T6	36
T10	T6	114
T11	T8	84
T12	T9	72
T13	T10 , T11	72
T14	T11	24
T15	T12	90
T16	T14	72
T17	T15	60
T18	T15	180
T19	T16	36
T20	T17	114
T21	T13 , T18	96
T22	T13 , T19	24
T23	T20	30
T24	T21	78
T25	T22	12
T26	T23 , T24	135
T27	T25	78
T28	T26	90
T29	T24 , T27	150
T30	T28 , T29	36

Tabla 1. Lista de actividades a realizar con las operaciones de precedentes y la duración en minutos de cada una de ellas.

Vale aclarar que el número mínimo teórico de estaciones está dado por la expresión (2).

$$k_{min} = \left\lceil {\sum_{j:tarea} t_j}/{TC} \right\rceil = \lceil 10.21 \rceil = 11, \text{ donde } TC \text{ es el tiempo de ciclo} \qquad (2)$$

La *solución inicial* que se considera es una solución propuesta en forma arbitraria factible que cuenta con 15 estaciones de trabajo, con tiempo de ciclo para esta configuración inicial de TC=198 minutos.

Estación	Tareas Asignadas	Tiempo Ocupado (min)	Tiempo Ocioso (min)
ET1	T2	144	102
ET2	T1,T3	192	54
ET3	T4,T6	246	-
ET4	T7	150	96
ET5	T8,T10	228	18
ET6	T5, T9, T11, T13, T14	240	6

Estación	Tareas Asignadas	Tiempo Ocupado (min)	Tiempo Ocioso (min)
ET7	T12, T15, T16	234	12
ET8	T17, T18	240	6
ET9	T19, T20, T21	246	0
ET10	T22, T23,T24, T25,T27	222	24
ET11	T26, T28	225	21
ET12	T29, T30	186	60

Tabla 2. Distribución de las tareas en las estaciones de trabajo obtenido con SA, tiempos ocupados de cada estacion y tiempo ocioso

Con esta solución inicial se aplicó la metaheurística de SA, con la cual se consiguen varias configuraciones alternativas con 12 estaciones, tiempo de ciclo de 246 minutos y eficiencia de línea del 86.48%. Una de ellas es la que se muestra en la Tabla 2.El tiempo de CPU incurrido para la resolución de este problema fue de 299.10 segundos encontrándose la solución reportada en la Tabla 2 en un total de 22157 iteraciones.

Si este mismo problema se resuelve mediante modelado matemático con la misma funcion objetivo, es decir, la minimización de las estaciones de trabajo utilizadas usando una versión del lenguaje GAMS win32 24.1.3, se llega a una solución alternativa a la anteriormente presentada, donde el tiempo de ciclo es de 246 min, la eficiencia de la linea es de 86.48% y el CPU incurrido es de 75.156 segundos.

Ejemplo de aplicación con 148 tareas

Se considera un proceso de manufactura compuesto de 148 operaciones, denotadas por T1, T2,…, T148. En la Tabla 3 se presenta la lista de las tareas, sus relaciones de precedencia y la duración en minutos de cada una de ellas. Se considera en este problema que el tiempo de ciclo es de 390 minutos.

Tareas	Tareas Predecesoras	Tiempo (min)	Tareas	Tareas Predecesoras	Tiempo (min)
T1	-	16	T75	T73	101
T2	-	30	T76	T73	5
T3	T2	7	T77	T74	28
T4	T3	47	T78	T74	8
T5	T3	29	T79	T74	281
T6	T1	8	T80	T75	7
T7	T1	39	T81	T80	26
T8	T4,T5	37	T82	-	10
T9	T6,T7	32	T83	T81,T82	21
T10	T8,T9	29	T84	T76	26
T11	-	17	T85	T83	20
T12	T10	11	T86	T84	21
T13	T10	32	T87	T77,T78	47
T14	T10,T11	15	T88	T79	23
T15	T12,T13,T14	53	T89	T88	13
T16	T15	53	T90	T88	19
T17	T16	8	T91	T87	115
T18	T17	24	T92	T89,T91	35
T19	T17	24	T93	T90	26
T20	T18,T19	8	T94	T92,T93	46
T21	T20	7	T95	T85,T86,T94	20
T22	T20	8	T96	T95	31
T23	T20	14	T97	T96	19
T24	T20	13	T98	T97	34
T25	T21	10	T99	T97	51
T26	T22	25	T100	T97	39
T27	T23	11	T101	T98	30
T28	T24	25	T102	T99	26
T29	T25,T26,T27, T28	11	T103	T99	13
			T104	T99,T100	45
T30	T29	29	T105	T104	58
T31	T30	25	T106	T104	28
T32	-	10	T107	T104	8
T33	-	14	T108	T103	383
T34	T32	41	T109	T101,T102	40
T35	T33	42	T110	T101,T102	34
T36	T31,T34,T35	47	T111	T109,T110	23
T37	T36	7	T112	T111	162
T38	T37	80	T113	T108	11
T39	T38	7	T114	T105,T106	19
T40	T39	41	T115	T107	14
T41	T37	47	T116	T112,T113	31

Tareas	Tareas Predecesoras	Tiempo (min)	Tareas	Tareas Predecesoras	Tiempo (min)
T42	-	16	T117	T115	32
T43	T42	32	T118	T114,T116, T117	26
T44	T43	66			
T45	T41	80	T119	T118	55
T46	T45	7	T120	T118	31
T47	T44,T46	41	T121	T118	32
T48	T40	13	T122	T120,T121	26
T49	T47	47	T123	T119	19
T50	T47	33	T124	T123	14
T51	T48,T49,T50	34	T125	T123	19
T52	-	11	T126	T122	48
T53	T51,T52	118	T127	T124	55
T54	T53	25	T128	-	8
T55	T54	7	T129	T124	11
T56	T54	28	T130	T125	27
T57	T54	12	T131	T126	18
T58	T55	52	T132	T126	36
T59	T55	14	T133	T131,T132	23
T60	T55	3	T134	T129,T130	20
T61	T56	3	T135	T127,T128	46
T62	T57	8	T136	T134,T135	64
T63	T57	16	T137	T133	22
T64	T61,T62	33	T138	T136,T137	15
T65	T63	8	T139	T138	34
T66	T58	18	T140	T139	22
T67	T59,T60	10	T141	T139	151
T68	T66,T67	14	T142	T139	148
T69	T64	28	T143	T140,T141	64
T70	T65,T69	11	T144	-	170
T71	T68,T70	118	T145	T144	137
T72	T71	25	T146	T142,T143	64
T73	T72	40	T147	T145,T146	78
T74	T72	40	T148	T147	78

Tabla 3. **Lista de actividades a realizar con sus operaciones de precedencia y su duración en minutos**

El número mínimo teórico de estaciones está dado por la expresión $k_{min} = \lceil 14.44 \rceil = 15$ estaciones.

La solución inicial que se considera es una solución factible, seleccionada en forma arbitraria. La misma cuenta con 22 estaciones de trabajo, con un tiempo de ciclo para esta

solución inicial de TC=383 minutos y eficiencia del 66,86%. A partir de esta solución inicial, se consiguen resultados con 16 estaciones de trabajo, un tiempo de ciclo de 390 minutos y una eficiencia de línea del 90,28%. La configuración factible resultante tras aplicar SA se representa en la Tabla 4.

Estación	Tareas Asignadas	Tiempo Ocupado (min)	Tiempo Ocioso (min)
ET1	T2, T3, T5, T11, T144, T145	390	-
ET2	T1, T4, T6, T7, T8, T9, T10, T12, T13, T14, T15, T16	372	18
ET3	T17, T18, T19, T20, T21, T22, T23, T24, T25, T26, T27, T28, T29, T30, T31, T42, T43, T44	356	34
ET4	T32, T33, T34, T35, T36, T37, T38, T41, T45	368	22
ET5	T39, T40, T46, T47, T48, T49, T50, T51, T52, T53, T54	377	13
ET6	T55, T56, T57, T58, T59, T60, T61, T62, T63, T64, T65, T66, T67, T68, T69, T70, T71	383	7
ET7	T72, T74, T79	346	44
ET8	T73, T75, T77, T78, T87, T88, T91	362	28
ET9	T76, T80, T81, T82, T83, T84, T85, T86, T89, T90, T92, T93, T94, T95, T96, T97, T98	379	11
ET10	T99, T100, T101, T102, T103, T104, T105, T106, T107, T109, T110, T128	380	10
ET11	T108	383	7
ET12	T111, T112, T113, T114, T115, T116, T117, T118, T119	373	17
ET13	T120, T121, T122, T123, T124, T125, T126, T127, T129, T130, T131, T132, T133, T134	379	11
ET14	T135, T136, T137, T138, T139, T141, T140	354	36
ET15	T142, T143	212	178
ET16	T146, T147, T148	220	170

Tabla 4. Distribución de las tareas en las estaciones de trabajo obtenido con SA, tiempos ocupados de cada estacion y tiempo ocioso

El tiempo de CPU incurrido para la resolucion de este problema a partir de este configuración inicial fue de 2643.9 segundos encontrándose la solución reportada en 2116 iteraciones.

Al querer resolver la minimización del número de las estaciones de trabajo mediante modelado matematico utilizando GAMS win32 24.1.3, se obtiene una solucion optima consistente en 16 estaciones con tiempo de ciclo de 390 minutos, pero en 5900 segundos de CPU, que es más del doble del tiempo consumido por la metaheuristica SA. Vale destacar que en el modelo matemático de balanceo la variable de decisión fundamental es la que asigna cada tarea a alguna estación de trabajo. Cuando se pretende encontrar la descomposición en estaciones, el número de estas variables a considerar es igual a n^2, donde n es el número de tareas que posee el proceso. En este último problema resulta entonces de 21904 variables.

4 CONCLUSIONES

Se desarrolló y aplicó la metaheurística del SA a problemas de líneas de ensamble para estudiar la minimización del número de estaciones de trabajo. La complejidad del problema está dada fundamentalmente por el número de tareas involucradas y su interacción a través del grafo de precedencias. El método resulta con muy buena performance tanto para problemas de mediano tamaño como para problemas de mayor tamaño. En el caso de los problemas relativamente pequeños, la estrategia de SA produce buenas soluciones (que nunca pueden catalogarse como óptimas) en tiempos que compiten con los requeridos por los modelos matemáticos para resolver el mismo problema. Sin embargo, cuando se pasa a problemas de mayor porte, el modelo matemático determina la optimicidad de las soluciones en tiempos de cómputo muy por encima de los requeridos por SA, el cual alcanza soluciones igualmente buenas en tiempos de cálculo moderados.

Si bien la metaheurística SA no permite garantizar la optimicidad de las soluciones halladas, el procedimiento aplicado es muy promisorio pues permite obtener soluciones alternativas cercanas al valor mínimo dado por el número teórico de estaciones, además no reporta una única solución sino que puede reportar varias soluciones alternativas, permite además poner una configuración inicial factible de partida, lo cual también influye en la

convergencia de la metaheurística, tiene gran facilidad de uso y su programación puede ser desarrollada en la mayoría de las plataformas de programación disponibles.

Estas características no se dan cuando se quiere resolver problemas de gran tamaño mediante programación matemática, ya que requiere una gran cantidad de recurso computacional.

APPLICATION OF THE "SIMULATED ANNEALING" METHEURISM TO THE BALANCING OF SIMPLE ASSEMBLY LINES

ABSTRACT: In general terms, a production and assembly line is a grouping of tasks in work stations that must respect certain precedence relationships. The resulting production system is very efficient for the mass production of parts, components and assemblies. One of the challenges of this production system is the so-called balancing, which consists of assigning the tasks to the work stations according to the production capacity and the productive time of the line, in such a way that the load of the stations is as balanced as possible. This work presents a positive experience in the resolution of the balancing of simple assembly lines (which manufacture a single product) using the metaheuristic "simulated annealing". In the literature, the application of other metaheuristics is frequently reported, such as taboo search and genetic algorithms to this type of problems, but very little the application of "simulated annealing", which motivated the interest to implement it. This metaheuristic in particular has characteristics that make it very attractive for its application to cases of real dimensions.

Keywords: Simulated Annealing, Simple Assembly Line, Assembly Line Balancing.

REFERENCIAS

DOLGUI, A.; PROTH, J. Supply-Chain Engineering. Editoral Springer (2010)

KIRKPATRICK, S., GELLAT JR, C.D., VECCHI, M.P. Optimization by Simulated Annealing. Science 220, 671-980 (1983)

LAPIERRE, S.D.; RUIZ, A.; SORIANO, P. "Balancing assembly lines with tabu search". European Journal of Operational Research 168, 826-837 (2006)

MATHWORKS, Matlab software, version 7.9.0 (R2009b), (2009).

MANAVIZADEH, N.; HOSSEINI, N.; RABBANI, M.; JOLAI, F. A Simulated Annealing algorithm for a mixed model assembly U-line balancing type-I problem considering human efficiency and Just-In-Time approach. Computers & Industrial Engineering, Volume 64, Issue 2, pp. 669-685 (2013).

REKIEK,B., DELCHAMBRE, A. The Balancing of Mixed-Model Hybrid Assembly Lines with Genetic Algorithms.13th ed. London: Springer-Verlag (2006)

SCHOLL, A.; BECKER, C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. European Journal of Operational Research 168, 666-693 (2006)

SCHOLL, A.; VOβ, S. Simple Assembly line Balancing- Heuristics approaches. Journal of Heuristics 2, pp. 217-244 · (1997)

Originais recebidos em: 04/12/2017

Aceito para publicação em: 19/12/2017