New Bounds for the Joint Replenishment Problem: Tighter, but not always better

Eric Porras*, Rommert Dekker
Econometric Institute, Tinbergen Institute, Erasmus University Rotterdam, P.O. Box 1738, 3000 DR Rotterdam, the Netherlands

Econometric Institute Report EI 2005-18

Abstract
In this paper we present new bounds on the basic cycle time for optimal methods to solve the JRP. They are tighter than the ones reported in Viswanathan [7]. We carry out extensive numerical experiments to compare them and to investigate the computational complexity.

Keywords: joint replenishment problem, bounds, computational complexity.

1. Introduction

Most of the optimal methods to solve the JRP presented in the literature are based on an algorithm first proposed by Goyal [2], which is based on enumeration of the total cost function between a lower and an upper bound of T. Van Eijs [1] proposed a modified version of Goyal's algorithm for cyclic strategies, where an explicit formula is introduced to obtain the intervals over which the total cost is enumerated. The pitfall of the algorithms by Goyal and van Eijs is that for large number of items and relatively high minor set-up costs, they require a large number of enumerations. Some people expect that this number increases exponentially in the number of items involved. Viswanathan [6] and Wildeman et al. [8] proposed the use of tighter bounds for the basic cycle time. Viswanathan [7] presented a comparative study of the performance of different methods until 2002. However, he did not consider the work by Wildeman et al. [8]. Thus, our objective is to perform a similar study to compare the Wildeman bounds with those from Viswanathan [6], and to investigate whether they can be combined to produce tighter bounds on T. Moreover, we also investigate the computational complexity of the algorithms. Several recent papers and text books present heuristics for the JRP ([3], [5]), but it is questionable whether they are necessary considering the speed of the exact algorithm presented in this paper.

2. Formulation

We consider the following formulation for the JRP:

* Corresponding author:
E-mail address: porrasmusalem@few.eur.nl

$$(P)\ \ Min\left\{TC(T,\mathbf{k})=\frac{S}{T}+\sum_{j=1}^{M}\left(\frac{s_j}{k_jT}+\frac{1}{2}h_jk_jD_jT\right)\ \middle|\ T>0, k_j\in Z^+, j=1,...,M\right\}$$

where **k** is the vector of the k_j's, D_j is the constant rate of demand for item j, T is the basic cycle time, S is the major set-up cost, and s_j and h_j are the ordering and holding costs of item j. Z^+ denotes the set of positive integers.

The function $TC(T,\mathbf{k})$ is not jointly convex with respect to T and **k**. However, for a fixed vector **k** the function $TC(T)$ is convex in T, with optimal T given by:

$$T^*(k_1,...,k_M)=\sqrt{\frac{2\left(S+\sum_{j=1}^{M}\frac{s_j}{k_j}\right)}{\sum_{j=1}^{M}h_jD_jk_j}} \tag{1}$$

Substituting (1) in TC gives the optimal cost for a given **k**:

$$TC(k_1,...,k_M)=\sqrt{2\left(S+\sum_{j=1}^{M}\frac{s_j}{k_j}\right)\left(\sum_{j=1}^{M}h_jD_jk_j\right)}$$

For given T, an optimal value of **k** is given in Wildeman et al. [8] by:

$$k_j(T)=\left\lceil -\frac{1}{2}+\frac{1}{2}\sqrt{1+\frac{8s_j}{h_jD_jT^2}}\right\rceil \quad \text{for } j=1,\ldots,M, \tag{2}$$

Now suppose we have an upper bound on T. We can use this as a starting value to enumerate the intervals with constant vectors **k**. For given $T^{(i-1)}$ and $\mathbf{k}^{(i-1)}$ we can determine the next break point $T^{(i)}$ from:

$$T^{(i)}=\max_j\left\{T_j^{(i)}\right\} \tag{3}$$

where

$$T_j^{(i)}=\sqrt{\frac{2s_j}{h_jD_jk_j^{(i-1)}(k_j^{(i-1)}+1)}} \quad \text{for } j=1,\ldots,M. \tag{4}$$

Notice that the optimal T associated with the vector $\mathbf{k}^{(i-1)}$, say $T^*_{(i-1)}$, as given by equation (1), does not necessarily belongs to the interval $[T^{(i)},T^{(i-1)})$. However, as stated in the following theorem, the overall optimal solution for TC has an associated optimal T, say T_{opt}, equal to some $T^*_{(i-1)}$, with corresponding optimal **k** given by $\mathbf{k}(T^*_{(i-1)})$.

Theorem 1. Let $\mathbf{k}_{\text{opt}}$ be the vector of k_j^* values that minimize the function $TC(T,\mathbf{k})$ among all possible T values as given by equation (5). Let $[T_{opt}^{(i)}, T_{opt}^{(i-1)})$ be the interval associated with $\mathbf{k}_{\text{opt}}$. Then $T_{opt} = T_{(i-1)}^*(k_1^*, k_2^*, ..., k_M^*) \in [T_{opt}^{(i)}, T_{opt}^{(i-1)})$.

Proof. See the Appendix.

In the next section we discuss the bounds used in Viswanathan's algorithm and the bounds suggested by Wildeman et al. [8], which were implemented in the algorithm proposed in this paper.

3. Bounds on *T* and solution method for problem (*P*)

As shown by van Eijs [1], an upper bound on T can be obtained from:

$$T_{upp}^{(1)} = \sqrt{\frac{2(S + \sum_{j=1}^{M} s_j)}{\sum_{j=1}^{M} h_j D_j}}$$

Note that for a large number of items and relatively high minor set-up costs, the previous upper bound can be very large. This increases considerably the computational effort to find the optimal TC. In Viswanathan [6] a tighter upper bound, denoted by $T_{upp}^{(V)}$, is obtained in the following way: start in $T_{upp}^{(1)}$ and use equations (1) and (2) recursively to find the first $T_{(i-1)}^*$ that lies inside its corresponding interval $[T^{(i)}, T^{(i-1)})$. The function TC will be monotone increasing between the overall optimal T and $T_{(i-1)}^*$.

Van Eijs [1] proposed a lower bound on T for cyclic policies as follows:

$$T_{low,VE} = 2S / TC^U$$

where TC^U is an upper bound on the total cost $TC(T,\mathbf{k})$, e.g. from applying $T_{upp}^{(1)}$.

This lower bound can be improved further by inserting in the last equation the best value of TC found so far in each step of the optimization algorithm. However, except for large values of the major set up cost and moderate minor set-up costs, the resulting lower bound can be very small. Starting with $T_{low,VE}$ Viswanathan [6] finds a tighter lower bound on T, say $T_{low}^{(V)}$, by using a similar procedure as the one described for $T_{upp}^{(V)}$. To avoid a large number of iterations to get the improved lower and upper bounds, Viswanathan stops the search if before reaching the best possible lower (upper) bound, the ratio $T_{upp}^{(V)} / T_{low}^{(V)}$ is below a predetermined value.

Wildeman et al. [8] used an entirely different approach to find bounds on T. They first obtained a lower envelope to the $TC(T)$ curve, say $TC^{(R)}$, by relaxing the integrality requirement of the k_j's. Then the procedure finds a locally optimal solution for the original function TC in $T = T(R)$, where $T(R)$ is the optimal solution of the relaxation (R). Then by determining the intersection between the level line corresponding to the feasible TC and the $TC^{(R)}$ curve, a lower bound on the interval $(0,T(R)]$ is obtained using bisection. It can be shown that the function $TC^{(R)}$ is convex in T [8] and therefore an upper bound on T, say $T_{upp}^{(W)}$, can be obtained by the same bisection procedure on the interval $[T(R),\ T_{upp}^{(1)}]$, whenever $T_{upp}^{(1)} > T_{upp}^{(W)}$, otherwise use $T_{upp}^{(1)}$. This procedure is summarized below:

First let $\phi_j(k_jT) = \dfrac{s_j}{k_jT} + \dfrac{1}{2}h_jD_jk_jT$ for $j = 1,\ldots, M$.

It is easy to verify that the function $\phi_j(k_jT)$ is strictly convex in T with a minimum for $T = x_j^*/k_j$, with $x_j^* = \sqrt{2s_j/(h_jD_j)}$. Now a lower bound on T, say $T_{low}^{(W)}$, is obtained from:

$$T_{low}^{(W)} = \frac{S}{TC(T(R),\mathbf{k}(T(R))) - \sum_{j=1}^{M}\left(\frac{s_j}{x_j^*} + \frac{1}{2}h_jD_jx_j^*\right)} \tag{5}$$

where $T(R)$ is the optimal basic cycle time for the relaxation (R) of problem (P):

$$(R)\ \min\left\{TC^{(R)}(T,\mathbf{k})\ \middle|\ T > 0,\ k_j \ge 1,\ j = 1,2,\ldots,M\right\}.$$

For the evaluation of $T(R)$ first assume w.o.l.g. that $x_1^* \le x_2^* \le \cdots \le x_M^*$ and denote by $h'(\cdot)$ the derivative of $TC^{(R)}$. Wildeman et al. [8] gives the following formula for $T(R)$:

$$T(R) = \sqrt{\frac{2(S + \sum_{j=1}^{j^*} s_j)}{\sum_{j=1}^{j^*} h_jD_j}} \tag{6}$$

where $j^* = \max\left\{\ 1 \le j \le M:\ h'(x_j^*) < 0\right\}$.

In Fig. 1 we show a graphical representation of the procedures to find the Viswanathan and Wildeman bounds.

Improved Wildeman bounds and optimization algorithms

We can improve the Wildeman lower bound in the following way: in each step of the optimization algorithm presented below check if the locally optimal value of $TC(T)$ is better than $TC(T(R),\mathbf{k}(T(R)))$, and whenever this is the case find a new $T_{low}^{(W)}$ by replacing in equation (5) $TC(T(R),\mathbf{k}(T(R)))$ with the best value of $TC(T)$. When we follow this improvement procedure the algorithm is called **Porras-Wild+**, otherwise it is referred to as **Porras-Wild**. Notice than the original Wildeman bounds can also be improved by using the same iterative procedure described for the Viswanathan bounds. This algorithm is referred to as **Porras-WV**. Finally, the algorithm with Viswanathan bounds is called **Visw**. For **Visw** and **Porras-WV** we stop the iterative procedure to improve the bounds when the ratio T_{upp}/T_{low} reaches the value 1.1.

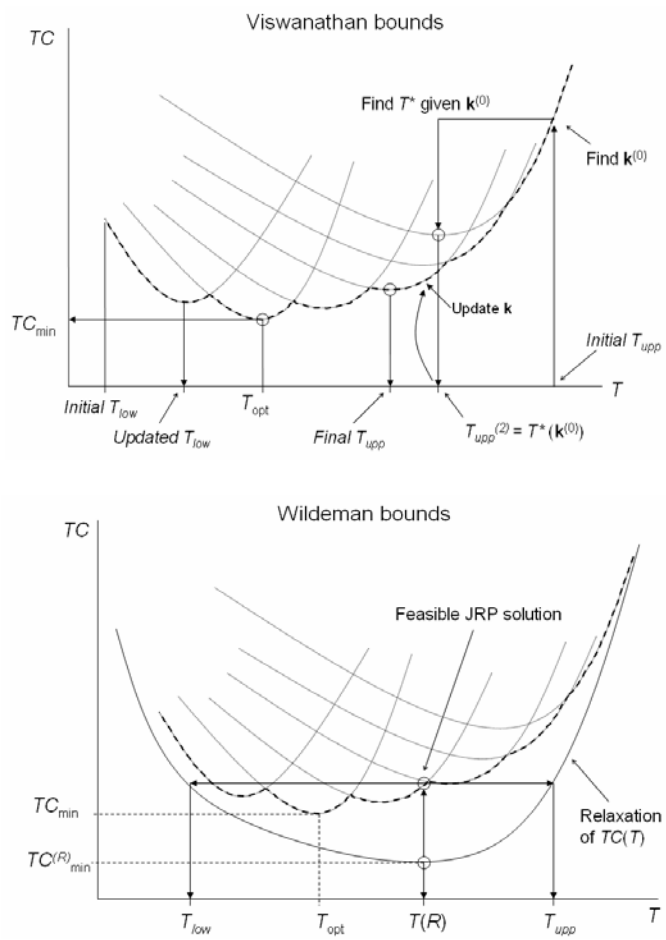

Fig.1: Graphical procedure to obtain the Viswanathan and the Wildeman bounds.

Below we formulate the complete algorithm, which is similar to van Eijs [1] but incorporates the result of Theorem 1 and different bounds on T.

Algorithm to solve (P)

Step 0. Initialization
Select BOUNDS = **Porras-Wild**, **Porras-Wild+**, **Porras-WV** or **Visw**.
Evaluate the bounds $T_{low}^{(\cdot)}$ and $T_{upp}^{(\cdot)}$ according to the selected BOUNDS.
Set $\mathbf{k}^{(0)} = \mathbf{k}(T_{upp}^{(\cdot)})$ using equation (2), $TC_{\min}^{(0)} = \infty$, $T^{(0)} = \infty$ and $n = 1$.
Evaluate $T_j^{(1)}$ for $j = 1, \ldots, M$ using formula (4).

Step 1. For $\mathbf{k}^{(n-1)}$ determine $T^{(n)}$ from (3) and set $J^{(n)} = \{ j\text{: } \max_j \{ T_j^{(n)} \} \}$.
Evaluate T_{n-1}^* using (1).

Set: $TC_{\min}^{(n)} = \begin{cases} \min\{TC_{\min}^{(n-1)}, TC(T_{n-1}^*, \mathbf{k}^{(n-1)})\} & \text{if } T_{n-1}^* \in \left[T^{(n)}, T^{(n-1)}\right] \\ \infty & \text{otherwise} \end{cases}$

Obtain the elements of the new vector $\mathbf{k}^{(n)}$, according to:

$$k_j^{(n)} = \begin{cases} k_j^{(n-1)} + 1 & \text{for } j \in J^{(n)} \\ k_j^{(n-1)} & \text{for } j \notin J^{(n)} \end{cases}$$

and set $T_j^{(n+1)} = \sqrt{\dfrac{2s_j}{h_j D_j k_j^{(n)} (k_j^{(n)} + 1)}}$ if $j \in J^{(n)}$. Otherwise $T_j^{(n+1)} = T_j^{(n)}$.

If BOUNDS = **Porras-Wild+** improve $T_{low}^{(W)}$ using (5) and $TC(T)$.

Step 2. If $T^{(n)} \leq T_{low}^{(\cdot)}$ STOP with $TC_{\min}(T, \mathbf{k}) = TC_{\min}^{(n)}$ and $T_{opt} = T_{n-1}^*$.
Otherwise set $n = n + 1$ and GOTO step 1.

END of the algorithm.

Computational complexity of the algorithms

Notice that in each step of the above algorithm the value of one ore more of the k_j's is increased by one, hence the maximum number of steps needed is given by:

$$Maximum\ \#\ of\ steps = \sum_{i=1}^{M} k_j(T_{low}) - k_j(T_{upp}) \tag{7}$$

For fixed T_{low} and T_{upp} this number increases linearly in the number of items, M. This has been unnoticed in the literature, as most papers give no explicit expression for the optimal k_j-values, like equation (2). Next assume that the initial list of $T_j^{(1)}$-values is sorted before entering *Step* (1) of the algorithm. Since the items change their k_j values one by one at each step of the algorithm with only one T_j-value updated in each round, it follows that the number of computation steps of the algorithm is $O(M \log M)$ under constant upper and lower bounds.

In the remainder of the complexity analysis, we need to set bounds on the s and hD values. This comes from a practical reason, since we assume that in reality there is always an effort associated with the handling or receiving of an item. Similarly, items are assumed to cause holding costs when kept on stock. Thus, for $s_j \in [s_{\min}, s_{\max}]$ and $h_j D_j \in [hD_{\min}, hD_{\max}]$ we distinguish the following cases:

a) S fixed.

For the Viswanathan bounds, first notice that $T_{low,VE}$ is proportional to $1/M$, since the total cost TC adds up M positive terms in s_j and h_jD_j, plus a constant term in S. By (2) it follows that $k_j(T_{low,VE})$ is proportional to M. Similarly, $k_j(T_{upp}^{(1)})$ is proportional to M. Now the iterative procedure of using equations (1) and (2) is linear in M, since the maximum number of k_j changes is given by (7) using T_{low}=$T_{low,VE}$ and T_{upp}=$T_{upp}^{(1)}$. It follows that under the Viswanathan bounds the algorithm has complexity $O(M^2 \log M)$.

For the Wildeman bounds, since we take the intersection of a relaxation of (P) with the TC curve, it follows from (6) that $T(R) \le T_{upp}^{(1)}$. Therefore:

$$TC(T(R), \mathbf{k}(T(R))) \le TC^U$$

and since the second term in the denominator of (5) is a positive constant it follows that $T_{low}^{(W)} > T_{low,VE}$. From this we have that after applying the Viswanathan iterative procedure to $T_{low}^{(W)}$ we get $T_{low}^{(W)+} \ge T_{low}^{(V)}$. From the preceding and using again the fact that TC adds M positive terms, we have from (5) that $T_{low}^{(W)}$ is proportional to $1/M$. Therefore, from (2) we have that $k_i(T_{low}^{(W)})$ is proportional to M. In a similar way it can be seen that $k_i(T_{upp}^{(W)})$ is proportional to M. From this it follows that the number of steps in the algorithm is proportional to $M^2 \cdot \log(M)$. Notice that the complexity to obtain $T(R)$ is $O(M \log M)$, since M x_j^*-values need to be sorted. Therefore the complexity of the overall algorithm remains $O(M^2 \log M)$ under Wildeman bounds.

b) S increases in M but M/S is bounded.

For the Viswanathan bounds we have that $T_{low,VE}$ remains bounded. Similarly the T*-values given by (1) remain bounded and therefore the number of steps in the iterative procedure to improve the bounds remain bounded as M increases. It follows that the maximum number of steps given above increases only linearly in M and thus the complexity of the algorithm is $O(M \log M)$.

For the Wildeman bounds we have that $T_{low}^{(W)}$ and $T_{upp}^{(W)}$ remain bounded as M increases. Therefore the number of steps in the algorithm increases linearly in M. It follows that the algorithm complexity is $O(M \log M)$.

For $S \downarrow 0$, the number of steps of the algorithm increases more than in the previous cases, however it is not such an interesting case since a practical lower bound on T can be used. Moreover, for small values of S the JRP is less relevant, and independent ordering for the items should be applied.

4. Computational results

We implemented the four algorithms presented previously using a similar experiment setting as Viswanathan [7], but with the inclusion of two extra values for the major set-up cost, so the values S = 0.5, 1, 5, 10, 20, 50, 100 were considered. The number of items considered were M = 10, 20, 50. For each value of S and M we generated 100 problems, with the minor set-up costs s_j and the unit holding costs h_j randomly generated from U[0.5,5] and U[0.2,2]. For each problem instance, demands for the individual items were randomly generated from U[100, 100000]. Therefore, a total of 7×3×100 = 2,100 problems were solved with each algorithm. In order to assess the effect of the number of items in the computational complexity of the algorithms, we also carried out extended experiments for M=250 (S=20, 25, 50, 100, 125), M=1000 (S= 20, 50, 100, 500) and M=5000 (S=100, 500, 2500).

In Table 1 we present a summary of the results for the four algorithms under consideration. The average number of intervals evaluated to get the optimal solution (including the ones needed to improve the bounds), the average lower bound, the average upper bound and the average CPU time is reported (including the computation time to obtain the solution of the relaxation (R)). We consider the latter as the performance criterion for the different algorithms.

Remark. We consider a ratio S/s_j varying from 0.1~1 until 20~200. From numerical results, we found that for ratios between 0.05 and 0.5 the JRP is still relevant with respect to independent EOQ ordering (savings up to 5%) (see Porras and Dekker [4]).

As we can see from Table 1, **Porras-Wild** and **Porras-Wild+** performed very similar, both dominating **Visw** in all problems solved except for M=5000 (S=100, 500, 2500), where the latter performs better. The reason is that for moderate number of items (M≤1000), the computation effort to improve the bounds in **Visw** consumes an important part of the overall time to find the optimal solution, together with the fact that the Wildeman procedure gives tighter initial bounds with less computational effort. This effect can also be appreciated in the results for **Porras-WV**, where the iterative procedure to improve the bounds increases the computation time considerably with respect to **Porras-Wild**. As the number of steps increases only linearly in M once the bounds are fixed, then for moderate M Wildeman bounds perform better. For M large (>1000) the effect is reversed, and the increment in the number of intervals in **Porras-Wild** becomes more relevant in the performance of the algorithm, thus **Visw** needs less CPU time. As for **Porras-WV**, it outperformed **Visw** in all problems solved, even for large number of items.

Finally, from the numerical results we can see that for M/S constant the CPU time for **Visw** and **Porras-Wild** remain bounded by a polynomial of $O(M \log M)$. Consider for example the results corresponding to M/S=10, 2 summarized in Table 2. We found an empirical bound for the CPU time of 1.2(c log c) in **Visw** and of 2(c log c) in **Porras-Wild**, where c is the increment factor in the number of items. From here we can see that for moderate M (<1000) **Porras-Wild** needs less computation time than **Visw**. The algorithms were implemented in Maple® v. 9.0 and ran using a Pentium 1.8 GHz processor.

Table 1. Comparison of JRP algorithms for determining the optimal cyclic policy with s_i~U[0.5,5]

		Average no. of intervals evaluated				*Average Tlow (years)*				*Average Tupp (years)*				*Average CPU time (sec.)*			
M	*S*	*Visw*	*Porras-Wild*	*Porras-Wild+*	*Porras-WV*	*Visw*	*Porras-Wild*	*Porras-Wild+*	*Porras-WV*	*Visw*	*Porras-Wild*	*Porras-Wild+*	*Porras-WV*	*Visw*	*Porras-Wild*	*Porras-Wild+*	*Porras-WV*
10	0.5	78.3	31.2	24.1	28.9	0.0022	0.0033	0.0038	0.0035	0.0074	0.0077	0.0077	0.0069	0.810	0.072	0.068	0.240
	1	49.1	21.0	18.0	17.6	0.0034	0.0043	0.0046	0.0048	0.0078	0.0082	0.0082	0.0074	0.627	0.063	0.060	0.202
	5	14.4	10.2	10.0	6.8	0.0080	0.0067	0.0068	0.0082	0.0093	0.0100	0.0100	0.0092	0.321	0.047	0.045	0.148
	10	8.0	8.0	7.8	4.1	0.0101	0.0078	0.0079	0.0101	0.0107	0.0113	0.0113	0.0106	0.218	0.045	0.044	0.122
	20	4.6	6.0	6.0	2.4	0.0122	0.0093	0.0093	0.0122	0.0125	0.0131	0.0131	0.0125	0.130	0.042	0.042	0.093
	50	2.4	4.6	4.6	1.3	0.0162	0.0119	0.0119	0.0162	0.0167	0.0170	0.0170	0.0167	0.075	0.040	0.040	0.073
	100	1.4	3.8	3.8	1.1	0.0214	0.0147	0.0147	0.0214	0.0217	0.0219	0.0219	0.0217	0.064	0.039	0.039	0.063
20	0.5	179.8	102.1	73.9	92.8	0.0018	0.0023	0.0028	0.0024	0.0069	0.0071	0.0071	0.0063	2.480	0.320	0.281	0.810
	1	113.5	66.6	53.8	58.6	0.0026	0.0031	0.0035	0.0033	0.0070	0.0076	0.0076	0.0067	2.009	0.198	0.174	0.571
	2	71.1	44.0	39.7	36.2	0.0037	0.0040	0.0043	0.0045	0.0073	0.0082	0.0082	0.0072	1.550	0.147	0.126	0.512
	5	34.3	27.0	25.2	20.2	0.0059	0.0055	0.0056	0.0066	0.0083	0.0090	0.0090	0.0082	0.988	0.138	0.118	0.468
	10	16.3	18.7	18.4	9.1	0.0084	0.0065	0.0066	0.0084	0.0092	0.0099	0.0099	0.0092	0.687	0.100	0.099	0.387
	20	9.0	14.0	14.0	4.6	0.0101	0.0078	0.0078	0.0101	0.0106	0.0113	0.0113	0.0105	0.449	0.092	0.092	0.294
	50	3.9	9.6	9.6	2.2	0.0129	0.0098	0.0098	0.0129	0.0133	0.0139	0.0139	0.0133	0.235	0.084	0.084	0.206
	100	2.0	7.2	7.2	1.3	0.0161	0.0118	0.0118	0.0161	0.0166	0.0170	0.0170	0.0166	0.134	0.081	0.081	0.125
50	0.5	532.2	440.3	287.7	374.8	0.0013	0.0014	0.0019	0.0015	0.0061	0.0063	0.0063	0.0056	13.400	3.780	2.850	6.100
	1	340.3	273.1	211.6	235.1	0.0019	0.0019	0.0023	0.0022	0.0062	0.0068	0.0068	0.0059	9.626	1.178	1.119	3.020
	5	100.4	106.1	97.8	70.4	0.0043	0.0038	0.0040	0.0045	0.0069	0.0080	0.0080	0.0069	4.878	0.546	0.505	2.124
	10	53.2	67.7	66.4	35.3	0.0060	0.0049	0.0051	0.0063	0.0076	0.0087	0.0087	0.0076	3.507	0.407	0.397	1.844
	20	23.0	48.9	48.4	14.9	0.0081	0.0060	0.0060	0.0081	0.0087	0.0095	0.0095	0.0087	2.247	0.336	0.331	1.357
	25	20.5	44.2	44.2	13.8	0.0085	0.0064	0.0064	0.0085	0.0092	0.0099	0.0099	0.0091	1.960	0.327	0.327	1.220
	50	10.8	30.0	29.9	7.7	0.0100	0.0076	0.0076	0.0100	0.0104	0.0112	0.0112	0.0104	1.240	0.293	0.293	0.820
	100	5.4	21.7	21.7	3.1	0.0120	0.0090	0.0090	0.0120	0.0123	0.0130	0.0130	0.0123	0.687	0.282	0.282	0.603
250	20	289.4	631.2	612.3	258.2	0.0044	0.0034	0.0035	0.0046	0.0063	0.0080	0.0080	0.0062	45.4	11.9	9.7	30.6
	25	248.1	528.0	521.0	231.9	0.0050	0.0038	0.0039	0.0051	0.0066	0.0082	0.0082	0.0066	39.2	10.2	8.0	26.8
	50	78.4	345.8	340.0	65.5	0.0071	0.0049	0.0050	0.0071	0.0078	0.0087	0.0087	0.0078	23.8	7.5	6.5	16.6
	100	44.7	226.5	224.0	36.8	0.0083	0.0060	0.0061	0.0083	0.0089	0.0095	0.0095	0.0089	14.0	5.8	5.4	10.3
	125	44.0	216.7	195.1	31.5	0.0086	0.0064	0.0064	0.0087	0.0093	0.0099	0.0099	0.0091	11.5	5.2	5.0	10.1
1000	20	1587.0	5287.2	5194.0	1486.0	0.0026	0.0019	0.0023	0.0028	0.0037	0.0062	0.0062	0.0037	536.0	432.8	420.7	441.1
	50	715.8	3410.0	3165.2	608.2	0.0042	0.0028	0.0030	0.0042	0.0050	0.0077	0.0077	0.0050	416.2	280.1	279.5	312.2
	100	484.2	2128.3	2166.5	407.0	0.0057	0.0038	0.0038	0.0057	0.0066	0.0082	0.0082	0.0066	263.7	152.2	151.8	200.8
	500	101.2	819.8	805.0	94.2	0.0087	0.0063	0.0063	0.0087	0.0094	0.0097	0.0097	0.0092	82.7	74.6	74.3	71.0
5000	100	3081.0	24230.0	22158.0	2948.4	0.0036	0.0026	0.0026	0.0037	0.0039	0.0048	0.0048	0.0039	5402.6	8735.0	8715.6	5190.8
	500	1140.8	10467.0	10174.0	1098.0	0.0061	0.0040	0.0040	0.0062	0.0067	0.0071	0.0071	0.0067	2336.1	3263.0	3260.0	2282.2
	2500	272.5	4265.8	4126.2	270.5	0.0087	0.0063	0.0063	0.0086	0.0092	0.0096	0.0096	0.0090	738.4	1718.2	1718.2	728.8

Table 2. Computational complexity of the algorithms

			Average no. of intervals		Average CPU time (sec.)	
M	*S*	*M/S*	*Visw*	*Porras-Wild*	*Visw*	*Porras-Wild*
10	1	10	49.1	21.0	0.627	0.063
	5	2	14.4	10.2	0.321	0.047
20	2	10	71.1	44.0	1.550	0.147
	10	2	16.3	18.7	0.687	0.100
50	5	10	100.4	106.1	4.878	0.546
	25	2	20.5	44.2	1.960	0.327
250	25	10	248.1	528.0	39.2	10.2
	125	2	44.0	216.7	11.5	5.2
1000	100	10	484.2	2128.3	263.7	152.2
	500	2	101.2	819.8	82.7	74.6
5000	500	10	1140.8	10467.0	2336.1	3263.0
	2500	2	272.5	4265.8	738.4	1718.2

5. Conclusions

In this paper we showed by numerical experiments that the bounds on T proposed by Wildeman [8] can be incorporated in an algorithm to solve the JRP that outperforms the best reported in Viswanathan [6] for a number of problem configurations, namely for moderate M. We show that this can happen in spite of the fact that Wildeman bounds are not always tighter than the latter. We also showed that the original Wildeman bounds can be further improved by two procedures, with **Porras-WV** resulting in tighter bounds than **Visw** for a number of problem configurations. Finally we showed that the JRP can be solved in $O(M^2 \log M)$ polynomial time, provided that the s_i and the h_iD_i remain bounded from below. Heuristics do not seem to be necessary for problems with less than 1000 items, since optimal methods can solve the JRP under mild conditions in polynomial time.

References

[1] M.J.G. van Eijs, A note on the joint replenishment problem under constant demand, Journal of the Operational Research Society 44 (1993) 185-191.

[2] S.K. Goyal, Determination of optimum packaging frequency of items jointly replenished, Management Science 21 (1974) 436-443.

[3] A.L. Olsen, An evolutionary algorithm to solve the joint replenishment problem using direct grouping, Computers & Industrial Engineering 48 (2005) 223-235.

[4] E. Porras, R. Dekker, Generalized solutions for the joint replenishment problem with correction factor, in: Report Series Econometric Institute, Erasmus University Rotterdam, EI 2005-19 (2005).

[5] E.A. Silver, D.F. Pyke, R. Peterson, Inventory Management and Production Planning and Scheduling, John Wiley & Sons (Eds.) (1998).

[6] S. Viswanathan, A new optimal algorithm for the Joint Replenishment Problem, Journal of the Operational Research Society 47 (1996) 936-944.

[7] S. Viswanathan, On optimal algorithms for the Joint Replenishment Problem, Journal of the Operational Research Society 53 (2002) 1286-1290.

[8] R.E. Wildeman, J.B.G. Frenk, R. Dekker, An efficient optimal solution method for the joint replenishment problem, European Journal of Operational Research 99 (1997) 433-444.

Appendix

Proof of Theorem 1

First note that from (2) it follows that $T^*(k_1,\ldots,k_M)$ is monotone decreasing in **k**. Now let $\mathbf{k}^{(i)}$ be the adjacent locally optimal vector to $\mathbf{k}_{\text{opt}}$ for $T > T_{opt}^{(i-1)}$ and suppose that $T^*(\mathbf{k}_{opt}) > T_{opt}^{(i-1)}$. By the convexity of $TC(T)$ it follows that TC is decreasing in $[T_{opt}^{(i)}, T_{opt}^{(i-1)})$ which implies that the minimum of TC is found in $T_{opt}^{(i-1)}$. It follows that $TC(T)$ is increasing for $T > T_{opt}^{(i-1)}$. Again by the convexity of TC this implies that $T^*(\mathbf{k}^{(i)}) < T_{opt}^{(i-1)} \Rightarrow T^*(\mathbf{k}^{(i)}) < T^*(\mathbf{k}_{opt})$, which is a contradiction by the monotonicity of T^*. Therefore, $T^*(\mathbf{k}_{opt}) < T_{opt}^{(i-1)}$ and the minimum of TC is to the left of $T_{opt}^{(i-1)}$. Proceed in a similar way to show that $T^*(\mathbf{k}_{opt}) \geq T_{opt}^{(i)}$, implying $T_{opt} = T^*(\mathbf{k}_{opt})$. □