HBBA: Hybrid Algorithm for Buffer Allocation in Tandem Production Lines

Alexandre DOLGUI [1,†], Anton V. EREMEEV [2], and Viatcheslav S. SIGAEV [3]

[1] *Division for Industrial Engineering and Computer Sciences, Ecole des Mines de Saint Etienne, FRANCE, dolgui@emse.fr*

[2] *Discrete Optimization Laboratory, Omsk Branch of Sobolev Institute of Mathematics, Omsk, RUSSIA, eremeev@ofim.oscsbras.ru*

[3] *Department of Mathematics, Omsk State University, Omsk, RUSSIA, SigVS@yandex.ru*

Abstract. In this paper, we consider the problem of buffer space allocation for a tandem production line with unreliable machines. This problem has various formulations all aiming to answer the question: how much buffer storage to allocate between the processing stations? Many authors use the knapsack-type formulation of this problem. We investigate the problem with a broader statement. The criterion depends on the average steady-state production rate of the line and the buffer equipment acquisition cost. We evaluate black-box complexity of this problem and propose a hybrid optimization algorithm (HBBA), combining the genetic and branch-and-bound approaches. HBBA is excellent in computational time. HBBA uses a Markov model aggregation technique for goal function evaluation. Nevertheless, HBBA is more general and can be used with other production rate evaluation techniques.

Keywords: Production line, Buffer allocation, NP completeness, Black-box complexity, Genetic algorithm, Branch-and-Bound method, Hybrid algorithm.

Submitted: May 12, 2005

Revised version: August 7, 2006

[†] **Corresponding author**: Dr. Alexandre Dolgui, Professor, Division Director and Department Head, Division for Industrial Engineering and Computer Science, Ecole des Mines de Saint Etienne, 158, Cours Fauriel, 42023 cedex France, FAX: +33 (0)4.77.42.66.66

1. Introduction and review of the literature

Buffer allocation problems arise in a wide range of applications: automatic transfer lines, production and assembly systems or communication networks. These exist in various forms, all aiming to answer the question: how much buffer storage to allocate between the processing stations? This question is important because the buffers may prevent blocking and/or starving of stations, thus drastically influencing the efficiency of the whole system. This is especially important for JIT environment which has as objective to reduce the inventory to as close to zero as possible. Therefore, we need to limit radically the stock by bounding the storage space between stations. In the context of Kanban policy, the buffer allocation problem is equivalent to the calculation of the minimal number of Kanbans.

An excellent illustration of the value to industry in solving problems of this type is given by A. Patchong, T. Lemoine and G. Kern (2003). The authors demonstrate how methods for buffer allocation in designing PSA Peugeot Citroën car-body shop yielded substantial profits. The practical importance of the optimization tools for buffers allocation was also demonstrated by Tempelmeier (2003). A detailed analysis of mathematical models describing the effect of the buffer storage may be found in the following books (Buzacott and Shanthikumar, 1993; Gershwin, 1994; Altiok, 1996) and comprehensive surveys (Buzacott and Hanifin, 1978; Dallery and Gershwin, 1992; Papadopoulos and Heavey, 1996).

In this paper, we consider a tandem production line (see Figure 1), where the parts are moved from one machine to the next by some kind of transfer mechanism. The

machines are subject to breakdowns: when a breakdown occurs, the corresponding machine is unusable for a random repair period. The machines are separated by finite buffers. The parts are stocked in these buffers when downstream machines are busy or down.

[Insert Figure 1 about here]

Let N denote the number of the intermediate buffers, and assume that the supply of new parts (raw materials) at the start of the line is inexhaustible and finished parts leave the machine N+1 immediately.

A machine is subject to failures only when it is operating. The failure and repair times are assumed mutually independent and exponentially distributed. Empirical studies indicate that this assumption is applicable in many cases – see e. g. (Inman, 1999). Let θ_b^i denote the mean time between failures of machine i, then $\lambda_i=1/\theta_b^i$ is its failure rate. Similarly, θ_r^i and $\mu_i=1/\theta_r^i$ are the mean time to repair and the repair rate of machine i, respectively, and i = 1, 2, ..., N+1.

We study tandem production lines where the machines have deterministic processing times (which are possibly non identical for different machines) – for automatic transfer and robotic assembly lines, this assumption is usually valid. Thus, machine i is assumed to have a constant cycle time θ_c^i and production rate $u_j=1/\theta_c^i$, i=1, 2,..., N+1.

The performance of this transfer line is measured in terms of production rate, i.e. the steady state average number of parts produced per unit time. For the evaluation of this

parameter, different types of Markov models have been considered in literature (see e.g. Dallery and Gershwin, 1992; Papadopoulos and Heavey, 1996).

In general, the production rate with finite buffers is difficult to analyze precisely with the Markov models. Exact performance computation of a production rate of a line with more than two machines and one buffer is problematic due to exponential growth of the number of states. Therefore, most of the techniques employed for the analysis of such systems are in the form of analytic approximations and simulations. Analytical approximations are generally based on the two-machines-one-buffer Markov models, and either aggregation (De Koster, 1987) or decomposition approach (Dallery *et al.*, 1989; Gershwin, 1987; Li, 2005). Simulation models are more expensive computationally but applicable to a wider class of systems (Dolgui and Svirin, 1995; Sörensen and Janssens, 2004).

In this paper, we use two-machines-one-buffer Markov model independently developed by Levin & Pasjko (1969), Dubois & Forestier (1982), and Coillard & Proth (1984). For each tentative buffer allocation decision, the production rate is evaluated via an aggregation algorithm (Dolgui, 1993; Dolgui and Svirin, 1995), which is similar to the Terracol and David (1987) techniques. This aggregation approach appears to be sufficiently rapid for evaluation of tentative buffer allocations within the optimization algorithms.

The aggregation algorithm for production rate evaluation consists in recurrent replacement of two adjacent machines by a single machine. The parameters λ^*, μ^*, u^* of the resulting single machine are calculated from differential equations corresponding to

the two-machines-one-buffer Markov model. After N steps of such aggregation procedure the system reduces to one virtual machine with parameters λ^*, μ^*, u^* and the estimate of the overall production rate $V(H)$ is given by $u^*\mu^*/(\lambda^*+\mu^*)$.

Note: the optimization algorithms proposed in this paper are general and can be used with other production rate evaluation techniques.

Let $H=(h_1, h_2, ..., h_N)\in Z^N$ be the vector of decision variables, where h_i is the size of the buffer between machines i and i+1. The problem of optimal buffer allocation has been considered in literature with respect to different optimality criteria (see e.g. Gershwin and Schor, 2000). The most commonly used among them are:

- Production rate $V(H)$;
- Total buffers capacity $B(H)=h_1+h_2+...+h_N$ or the cost of buffer equipment (linear in H);
- Average steady state inventory cost $Q(H)= c_1q_1(H)+ ...+c_N\, q_N(H)$, where $q_i(H)$ is the average steady state number of parts in buffer i, for i=1, 2, ..., N.

For example, Yamashita and Altiok (1998) suggest a dynamic programming approach for the minimization of total buffer space when the required value of production rate $V(H)$ is given as a constraint. In (Jafari and Shanthikumar, 1989) the dynamic programming is used to maximize $V(H)$ given a total buffer capacity. For similar knapsack problem formulations, Vouros & Papadopoulos (1998) suggest a knowledge-based system, and Gershwin & Schor (2000) present gradient-based methods (here both discrete and continuous flows of material are considered). In (Spinellis and Papadopoulos, 2000), a genetic heuristic and a simulated annealing algorithm are

developed. Shi and Men (2003) present a hybrid Nested Partitions and a Tabu Search algorithms for maximizing $V(H)$ under the constraint of a total buffer capacity. An original criterion is put forward by Helber (2001): buffer space allocation is considered as an investment problem. A gradient algorithm is tested to determine the buffer allocation that maximizes the expected net present value of the investment, including machines, buffers and inventory.

The optimization method HBBA offered in this paper is based on a branch-and-bound algorithm (BBA) that uses an initial approximate solution found by a genetic algorithm (GA) analogous to the GA developed by Dolgui *et al.* (2002).

Note: The GA for finding the initial approximate solution was chosen after testing various other versions of GAs and several tabu search (TS) algorithms we developed. One of the most efficient versions of the TS algorithm was one which used a random neighborhood space and constant length of tabu-list – see e.g. Glover and Laguna (1997). However, the selected GA achieved better results. The superiority of the GA vs. TS is partly explained by the effects of a population that allows the GA to adaptively search in different areas of the decision space. This feature proved to be helpful in the Nested Partitions algorithm of Shi and Men (2003), which is also based on information accumulation helping to concentrate the search in the most promising areas.

Taking into account that the algorithm HBBA is coupled with an approximate production rate evaluation algorithm, it becomes an approximation algorithm as well. Nevertheless, the precision of the HBBA is provably close to that of goal function evaluation.

2 Optimization problem properties

2.1 Criterion

Let us introduce the following additional notation:

T_{am}	amortization time of the line (line life cycle);
$R(V)$	revenue for the sold production per time unit;
$J(H)$	buffers acquisition cost for configuration H;
d_i	maximal admissible size for buffer i, $i=1, 2,\ldots, N$.

In this paper, we deal with the following criterion:

$$Max\ \varphi(H)=T_{am}\, R(V(H)) - J(H). \qquad (1)$$

The functions $R(V)$ and $J(H)$ are assumed monotone and non-decreasing. These functions may incorporate some penalties, fixed costs for different standard buffer sizes, overproduction price reduction, etc. Function $\varphi(H)$ is to be maximized subject to the constraints $h_1 \le d_1, h_2 \le d_2, \ldots, h_N \le d_N$.

2.2 Problem complexity

In Appendix, we give a proof of NP-hardness for a simple case of parallel-serial lines. The problem considered in the main body of this paper is more difficult to analyze, since it contains two arbitrary non-decreasing functions $R(V)$ and $J(H)$, which makes the usual complexity analysis (as e.g. in Garey and Johnson, 1979) not quite adequate.

Therefore, we analyzed the complexity of the problem in terms of black-box optimization (see e.g. Droste *et al.*, 2006). Black-box optimization is used when we do not have an access to the specific parameters of the given instance but may collect information about the unknown parameters only through goal function evaluations.

Let us call the number of tentative solutions examined by a search algorithm (randomized or deterministic), until the optimum is found, *the optimization time*. The complexity of a black-box optimization algorithm is defined as the expected (average) optimization time for the worst-case instance, which is a function of the problem size. This approach is well suited for analysis of the problem hardness for a wide class of modern heuristic methods, such as genetic algorithms, simulated annealing, evolutionary strategies, tabu search, etc. (see e.g. Reeves, 1993), where the search process is mainly directed by the goal function values of already visited solutions.

Proposition 1. *For the buffer space allocation problem for a line consisting of two machines separated by a finite buffer, the expected optimization time of any black-box optimization algorithm is lower bounded by* $d_1/2+1$.

Proof. Let the integer upper bound d_1 for the buffer size h_1 be given. In order to construct a hard case for optimization, one can define such functions $R(V)$ and $J(h_1)$ for

this line that $\varphi(h_1)$ will be constant (let it be 0) for all integer values of h_1, except at one point, where $\varphi(h_1)$ takes its maximal value (assume that here $\varphi(h_1)$ equals 1). This is based on the fact that function $V(h_1)$ is strictly increasing on $[0,d_1]$ (this can be shown using the closed-form expressions e.g. from Coillard and Proth, 1984), so the maximum of $\varphi(h_1)$ may be "hidden" in any point $0,1,\ldots,d_1$.

Now we can apply the Yao's minimax principle (see e.g. Motwani and Raghavan, 1995), which gives the lower bounds on complexity of the randomized black-box optimization algorithms through the analysis of deterministic algorithms for the same problem. Let us limit the set of problem inputs to those where $\varphi(h_1)$ takes only values 0 and 1 for h_1 in $\{0,1,\ldots,d_1\}$ (this will further simplify the problem). In our situation, for any fixed value of d_1, the Yao's minimax principle implies that the optimization time of any black-box algorithm for its worst-case function $\varphi(h_1)$ is lower-bounded by the expected time of the worst-case optimal deterministic black-box algorithm (for the same value of d_1) working on the inputs with any given probability distribution of function $\varphi(h_1)$.

Taking $\varphi(h_1)$ uniformly distributed on the set of functions with a single maximum (equal to 1) in $\{0,1,\ldots,d_1\}$, we conclude that the expected optimization time of any worst-case optimal deterministic black-box algorithm is $d_1/2+1$. Thus, by the Yao's minimax principle, the expected optimization time of any black-box algorithm for our problem is lower bounded by $d_1/2+1$ (the obtained lower bound is tight, as it follows Proposition 2 from Droste *et al.*, 2006). □

Finally, our Proposition 1 demonstrates that the black-box complexity of buffer space allocation problem can be arbitrarily large as we increase the maximal admissible buffer size d_1. In addition, this complexity increases drastically with the number of buffers in line.

3 Optimization method

The overall approach of the method HBBA is presented in Figure 2.

[Insert Figure 2 about here]

We use the standard depth-first branching procedure for the BBA. This routine will be employed in two different ways: as a single-dimension (one buffer only) optimizer inside the GA, then it is denoted BBA_1, or as a separate algorithm for solving the full-scale problem (all buffers), then it is denoted BBA_N.

To describe the hybrid optimization algorithm HBBA, we start with the BBA procedure used.

3.1 Branch-and-bound algorithm (BBA)

In our BBA, a node of the branching tree is a 5-tuple (*F, a, b, g, j*), where $F \subset \{1,2,\ldots,N\}$ is a set of fixed coordinates (buffers for which their size is fixed); $j \in \{1, 2,\ldots, N\} \backslash F$ is the index of a buffer with non fixed size bounded between *a* and *b*; $g \in Z^{|F|}$ is a vector containing the fixed values of coordinates (buffer sizes) for buffers with indices in *F*.

Let us define a set $D = \{H \in Z^N : h_i \in [0, d_i],\ i = 1, 2, \ldots, N\}$. So, each 5-tuple is associated with a set of solutions:

$$S(F, a, b, g, j) = \{H \in D : h_i = g_i, i \in F; h_j \in [a, b]\}.$$

Branching at node (*F, a, b, g, j*) is performed as follows:

If $a \neq b$, then the associated subset *S*(*F, a, b, g, j*) splits into

$$S(F, a, \frac{a + b}{2}, g, j) \text{ and } S(F, \frac{a+b}{2} + 1, b, g, j).$$

Otherwise (if $a = b$), it divides into

$$S(F \cup j, 0, \frac{d_{j+1}}{2}, g, j+1)$$

and

$$S(F \cup j, \frac{d_{j+1}}{2} + 1,\ d_{j+1}, g, j+1).$$

The upper bound (*UB*) on the goal function value for the set of the solutions *S*(*F, a, b, g, j*) is given by

$$UB = \psi(S(F, a, b, g, j)) = T_{am} R(V(H^{\max})) - J(H^{\min}), \qquad (2)$$

where

$$h_i^{\max} = g_i\,,\ h_i^{\min} = g_i\,,\ i \in F\ ,$$

$$h_j^{\max} = b\,,\ h_j^{\min} = a\,,$$

$$h_k^{\max} = d_k\,,\ h_k^{\min} = 0\,,\ k \notin F\ ,\ k \neq j\ .$$

In particular, for a leaf node (complete solution) in the branching tree, this bound coincides with $\varphi(H)$, where $H = H^{\max} = H^{\min}$ is the only element of the leaf node.

Validity of this *UB* follows from the fact that *V(H)* is an increasing function, i.e. for any $i \in \{1,\ 2,\ \ldots,\ N\}$, given arbitrary fixed capacities $h_1,\ldots,\ h_{i-1},\ h_{i+1},\ldots,\ h_N$, *V(H)* is increasing as a function of h_i. This fact was proved using the sample path approach (see e.g. Glasserman & Yao, 1996 or Buzacott & Shanthikumar, 1993, Chapter 6.6).

The initial lower bound *LB* is assumed to be -∞, if it is not given explicitly in the input of the BBA. In what follows, by "Pure BBA" we mean BBA_N started from node $(\varnothing, 0, d_1, \varnothing, 1)$ with $LB = -\infty$.

3.2 Local optimization heuristic

This local optimization heuristic based on the BBA is used in the GA below – we denote it by $LO_{BBA}(H)$. This procedure aims to improve a given solution H with the help of the BBA_1 applied to one buffer at a time, while the other buffers are fixed:

Algorithm $LO_{BBA}(H)$

1. Generate a random permutation $(\pi_1, \pi_2, ..., \pi_N)$ of elements $\{1, 2,..., N\}$.

2. For all i from 1 to N do:

   2.1 Set $F_i := \{ \pi_1, ... \pi_{i-1}, \pi_{i+1}, ... , \pi_N \}$; $j := \pi_i$;

   $g_t := h_t$ for all $t \neq i$.

   2.2 Start BBA_1 from node $(F_i, 0, d_j, g, j)$. Let H' be the output of BBA_1.

   2.3 If $\varphi(H) < \varphi(H')$ then set $H := H'$.

3. Return H.

GA using LO_{BBA}. The general scheme of the GA coincides with that of the genetic algorithm proposed in (Dolgui *et al.*, 2002), except for the local optimization procedure. A solution (individual) is presented by a vector of components (genes), where each gene gives the size of the corresponding buffer. The solutions of the initial population are randomly generated according to an a priori defined probability distribution. At each iteration, a couple of new solutions are obtained, and they replace a couple of "unpromising" solutions chosen in the current population (thus the population size remains constant). The best solution is returned when the GA stops.

Construction of a new pair of solutions (offspring) starts with choosing a pair of parents from the current population by the means of a probabilistic *selection* operator. We use the s-tournament selection operator which randomly chooses s individuals from the

current population and selects the best one as a parent. Then the standard one-point crossover operator (see e.g. Reeves, 1993) replaces some coordinates (genes) of one parent with the values taken from the corresponding positions of the other. This is done with a fixed crossover probability P_{cross} (otherwise the crossover has no effect). After crossover, the strings undergo *mutation*, where the genes are randomly altered: the size of each buffer receives a uniformly distributed random variation of not more than Δ units (parameter Δ is chosen experimentally before the run of the GA).

The local optimization heuristic LO_{BBA} is applied to the obtained solutions before they are added into population. This modification of the GA using the LO_{BBA} has turned out to be more advantageous than those suggested in (Dolgui *et al.*, 2002).

3.3 The hybrid algorithm HBBA

The hybrid algorithm HBBA consists in starting the GA before the BBA for finding an approximate solution (see Figure 2). This solution is used further in the BBA, defining an initial lower bound (*LB*). If no improving solutions are found in the BBA, then this solution is returned as a result of the HBBA:

Algorithm HBBA

1. Obtain solution H using GA with LO_{BBA}.
2. Start BBA_N from node $(\varnothing,0,d_1,\varnothing,1)$ with $LB=\varphi(H)$.
3. If BBA_N finds H', $\varphi(H)<\varphi(H')$ then return H',

   otherwise return H.

As mentioned before, the BBA is used in HBBA both for solving the one-dimensional sub-problems in the local optimization procedure of GA and for the full-scale problem after running the GA.

3.4 Some remarks on precision of BBA

To implement the BBA with such bounds we need to compute the functions $\psi(S(F,a,b,g,j))$ and $\varphi(H)$ exactly, which is problematic, since no exact method is known to evaluate the production rate $V(H)$. Instead, let us introduce the functions $\Gamma(S(F,a,b,g,j))$ and $\Phi(H)$ defined by analogy with $\psi(S(F,a,b,g,j))$ and $\varphi(H)$, respectively, except for the production rate V which is now substituted by its approximate value, computed via a Markov-model aggregation heuristic.

We can formulate an a priori precision of the BBA in terms of deviations of $\Gamma(S(F,a,b,g,j))$ from $\psi(S(F,a,b,g,j))$ and $\Phi(H)$ from $\varphi(H)$. Let us consider a general case and assume that for the problem being considered, the values ε and Δ are such that

$$|\varphi(H) - \Phi(H)| \le \varepsilon \text{ for all } H \in D,$$

and for any 5-tuple (*F, a, b, g, j*) holds

$$\Gamma(S(F,a,b,g,j)) \ge \psi(S(F,a,b,g,j)) + \Delta.$$

The following simple proposition bounds the precision of the BBA.

Proposition 2. *If H^* is an optimal solution and H' is the output of the BBA, then*

$$\varphi(H^*) - \varphi(H') \le \varepsilon + \max(\varepsilon, \Delta).$$

Proof. Let us assume $H^* \neq H'$. This means that one of the pruned nodes of the branching tree was associated with a set of solutions $S(F,a,b,g,j)$, containing the solution H^*. If this node is a leaf, then $\Phi(H') - \Phi(H^*) \ge 0$, otherwise

$$\Phi(H') - \Gamma(S(F,a,b,g,j)) \ge 0.$$

In the first case we have

$$\varphi(H') + \varepsilon - (\varphi(H^*) - \varepsilon) > 0,$$

$$\varphi(H^*) - \varphi(H') \le 2\varepsilon,$$

and in the second case

$$\varphi(H') + \varepsilon - (\varphi(H^*) - \Delta) > \Phi(H') - \Gamma(S(f,a,b,g,j)) \ge 0,$$

therefore,

$$\varphi(H^*) - \varphi(H') \le \varepsilon + \Delta. \qquad \square$$

In our case $\Delta = \varepsilon$, so $\varphi(H^*)-\varphi(H') \leq 2\varepsilon$. Note: the complete enumeration of all feasible solutions in the worst case also yields an error 2ε. Thus, the BBA achieves the best possible precision in some sense (it does not introduce any additional errors).

4. Computational experiment

The described algorithms were programmed in Delphi 6.0 and tested on a computer with Celeron 1,7GHz processor, 128 Mb RAM.

4.1 Randomly generated tests

In this part of experiments, we used 24 randomly generated series of tests listed in Table 1.

[Insert Table 1 about here]

Each series in Table 1 includes 30 different instances (different lines with the same following parameters: number of buffers and maximal buffer size). Here in the notation of each series **G*N*_*m*** and **W*N*_*m***, index N is the number of buffers in the corresponding lines and m is the upper limit on the admissible buffers size, i.e. $d_i=m$ for all i=1, 2,…, N.

The HBBA was compared to the Pure BBA in terms of the running time and the solutions obtained (both algorithms were executed until the branching was finished). The GA with LO_{BBA} here (used in HBBA) has the same settings of the internal parameters as the GAs in (Dolgui *et al.*, 2002): population size 50, tournament size s=5,

maximal admissible variation of each buffer size in mutation Δ=5, P_{cross}=0.5 and stops after 1000 iterations.

Some instances with relatively small cardinality of solutions space were solved by the complete enumeration method (CEM). In what follows, we call the GA with LO_{BBA} by "GA" for short. We denote the average running times of the HBBA (including Step 1, i.e. the work of the GA), Pure BBA, CEM, and GA (only Step 1 of HBBA) by t_{HBBA}, t_{BBA}, t_{CEM} and t_{GA}, respectively. These times are measured in seconds. Each algorithm is run once per instance.

The instances (lines) of series **G*N*_*m*** were generated with the following parameters: for all i=1, 2, ..., N+1 we set u_i=1 and choose $\mu_i \in$[1,100], $\lambda_i \in$[1,100] with uniform distribution. For all of these lines the buffer acquisition cost $J(H)$ is equal to 10*$B(H)$, and the amortization time T_{am}= 7000. The revenue is $R(V(H))$ = 10*$V(H)$.

The random series **W8_*m***, m=5, 10, 15, 20, 25 consist of the lines, where the value of buffer sizes in the optimal solution is close to the maximum size. Here N=8, and for all i=1, 2, ..., N+1 we set u_i=1 and choose $\mu_i \in$[10,12], $\lambda_i \in$[11,13] with uniform distribution. Other parameters are the same as in the series described above.

In Table 2, we show the minimum $t^{\min}_{HBBA}$, average t_{HBBA} and maximum $t^{\max}_{HBBA}$ running time of the HBBA as well as the average running time of the CEM and the GA for each test series **G*N*_*m***.

To simplify future comparisons with other algorithms, the average number of tentative solutions evaluated in the HBBA, including those computed for finding the BBA bounds (2), is given in column *Sol*.

Here for series from **G5_20** to **G5_35** and for **G6_20** we indicate the actual running time of the CEM. For the other test series, the time is estimated using the total number of elements in the space of solutions as

$$t_{CEM} \approx \tau_0 \prod_{i=1}^{N} (d_i + 1),$$

where τ_0 is the time required for a single evaluation of the goal function (1).

[Insert Table 2 about here]

We were unable to compare the proposed algorithms with the CEM on the whole set of test series due to the immense computational time required for the CEM for the larger problems. However, it turned out that for the series from **G5_20** to **G5_35** and for **G6_20**, the goal function values of solutions returned by the HBBA were identical to those of the solutions obtained by the CEM. Table 2 shows that the running time of the HBBA is much smaller than the CEM and this advantage increases with the growth of the number and size of buffers.

In order to evaluate the efficiency of hybridization we have compared the running time of the Pure BBA to the time of the branch-and-bound-phase (Step 2) of the HBBA. The time of the GA-phase (Step 1) of the HBBA is neglected here because usually it takes

about one second, as seen from Table 2. In Figure 3 and Figure 4, we give the speed-up ratio computed using the running times as

$$\frac{1}{30}\sum_{i=1}^{30}\frac{\tau_{HBBA}{}^{i}}{\tau_{BBA}{}^{i}},$$

where $\tau_{BBA}{}^{i}$ denotes the running time of the Pure BBA and $\tau_{HBBA}{}^{i}$ is the running time of the BBA-phase (Step 2) of the HBBA for the problem number *i* in a particular series. Note that we used 30 instances in each series, so *i*=1,2,…,30. The confidence intervals correspond to 95% level. The results for other series have the same sort of behavior. For series **G*N*_*m***, the speed-up factor is always present on these problems and it approaches the ratio 0.6 approximately (see Figure 3). For series **W8_*m*,** with the growth of problem size, the acceleration becomes more significant, approaching 0.06 (see Figure 4).

[Insert Figure 3 about here]

[Insert Figure 4 about here]

4.2 Known knapsack-type tests

Many publications focus on knapsack formulations of buffer space allocation problem. Therefore, we also tested our algorithms on two series of 5-machine knapsack-type problems **vp6.3 - vp6.10** and **vp7.3 - vp7.10** which were suggested by Vouros and Papadopoulos (1998). In their paper, the overall amount of buffer space was limited

from above by the value given in the problem index (i.e. for k = 3, 4, …, 10, the set of admissible solutions for **vp6.*k*** and **vp7.*k*** is restricted by the condition $B(H) \le k$). The maximization criterion is the output rate $V(H)$.

In order to take into account the knapsack-type constraint in our case, we have defined $\varphi(H)$ combining the output rate with a linear penalty: $\varphi(H)=V(H)-10000\times\max\{0,B(H)-k\}$. For all of these test examples, we set the buffer acquisition cost $J(H)$ equal to $B(H)$ and the amortization time T_{am}=1, so the revenue is $R(V(H)) = V(H)$. The parameters of machines in series **vp6.3 - vp6.10** were the following: $\mu_1= \mu_2= \mu_3= \mu_4= \mu_5= 0.5$, $\lambda_1= 0.1$, $\lambda_2= 0.2$, $\lambda_3= 0.25$, $\lambda_4= 0.3$, $\lambda_5= 0.35$, $u_1= u_2= u_3= u_4= u_5= 1$. Here we set the production rates u_1, u_2, …, u_5 equal to the corresponding mean production rates in (Vouros and Papadopoulos, 1998). Similarly we assign the following parameters of machines in series **vp7.3 - vp7.10**: $\mu_1= \mu_2= \mu_3= \mu_4= \mu_5= 0.5$, $\lambda_1= \lambda_2= \lambda_3= \lambda_4= \lambda_5= 0.05$, $u_1= 1$, $u_2= 1.1$, $u_3= 1.2$, $u_4= 1.3$, and $u_5= 1.4$.

In Table 3, we show the solutions for **vp6.3-vp6.10** and **vp7.3 - vp7.10** obtained by the Pure BBA (column H), their goal function values (column φ), and the corresponding running time.

[Insert Table 3 about here]

Note that the BBA time for considered knapsack-type problems is very short (<0.2 sec.), whereas the average running time of the GA for all of these problems was relatively large (about 0.55 sec.), therefore the usage of the hybrid scheme obviously does not make sense here.

5. Discussion of generalizations

A natural generalization of the buffer space allocation problem considered in this paper is the extension to the production lines with series-parallel structure. The line can be represented by a series-parallel digraph, where the machines correspond to arcs and the buffers correspond to nodes. The genetic algorithm (coupled with aggregation techniques for performance analysis) discussed above is also applicable to these lines (Dolgui *et al.*, 2002).

However, the proposed BBA is harder to generalize, because the series-parallel lines are significantly harder with respect to finding appropriate bounds. The *UB* used in this paper is based on monotonicity of the function $V(H)$ which was proved by the sample path approach. But, for the series-parallel lines, we found a counterexample with 4 machines (see Figure 5), which shows that the sample path approach is not applicable for the proof of monotonicity of function $V(H)$.

[Insert Figure 5 about here]

In this counterexample, we use the following assumptions:

- A part goes from Buffer 1 to Machine 2, if Machine 2 is free, otherwise it goes to Machine 4,
- all machines never fail,
- $u_1 = u_2 = u_4 = 1$, $u_3 = 1/2$,
- the last buffer is inexhaustible.

Let compare two following examples. Let us assume that the sizes of the first and second buffers in the first case are zero: $h_1 = h_2 = 0$, and in the second case: $h_1 = 0$, $h_2 = 1$.

In Tables 4 and 5, we list the timetable for the movements of parts. The part arrival times are given in column "A" and part departure times are given in column "D".

[Insert Table 4 about here]

[Insert Table 5 about here]

The tables show that the number of parts processed by the line till time 6 in the first case (4 parts) is greater than in the second one (3 parts), so the monotonicity property does not hold.

Of course this example does not imply that the monotonicity of $V(H)$ can not be established by some other method, different from the sample path. Nevertheless, using the simulation (see Dolgui, 1993 for details of the simulation algorithm) on the line with the same structure but with other parameters we found that $V(H)$ increases when h_2 is reduced (with statistical significance level 0.05). The parameters that we used in this experiment were $\lambda_i = \mu_i = 1$ for all $i = 1,2,3,4$, $u_1 = u_2 = 1$, $u_3 = 0.5$, $u_4 = 2$, $h_1 = 10000$, $h_2 = 50$, h_3 equals 10, 11 and 20, $h_4 = \infty$.

6. Conclusions

A buffer allocation problem for unreliable tandem lines was considered. We suggested a method named HBBA. This method has a precision comparable to that of the complete enumeration. HBBA is based on the branch-and-bound approach, complemented by a genetic algorithm for finding initial solutions. For the evaluation of production rate of tentative design decisions we used a Markov-model aggregation method, however our optimization technique is quite general and it may be used for other evaluation methods and models, provided that $V(H)$ is a non-decreasing function of H.

The computational experiments showed that the usage of an initial solution, obtained by the genetic algorithm, in the branch and bound procedure, can shorten the total running time as compared to the pure branch-and-bound algorithm. Another new feature of the proposed hybrid algorithm HBBA – the insertion of the one-dimensional Branch and Bound algorithm into the genetic algorithm for local optimization – turned out to be advantageous.

Further research might address the production lines with series-parallel structure. An efficient upper bound for this type of line needs to be found. Moreover, it would be interesting to use the statistical information collected in the genetic algorithm for finding the order of branching that suits a given problem.

Appendix

It would be worthwhile to investigate the complexity of buffer space allocation problem when the goal function is not so broad. For example, many authors use the knapsack-type formulation – in our notation it corresponds to the assumption that $J(H)$ is linear, and if $V(H)$ is above a given threshold V^0, then $R(V(H))$ equals to some sufficiently large constant, $R(V(H))=0$ otherwise. We were not able to obtain any

stronger results in this direction for the case of tandem production lines considered in this paper; however the knapsack-type problem for series-parallel lines turned out to be *NP*-hard. To proving this, we need to have a procedure that computes the production rate in time bounded by a polynomial in length of problem input data. It will be sufficient for us to present such a procedure only for the production systems with *simple structure*. By system with a simple structure, we mean a system consisting of parallel chains only, where each chain has at most two sequential machines, there are no links between the chains, and all chains start at Machine 1 and terminate at the last machine.

Proposition 3. *The problem of finding the buffer space allocation vector* $H=(h_1, h_2,..., h_{n-1}) \in Z_+^{n-1}$ *minimizing the criterion* $\sum_{j=1}^{n-1} b_j h_j$ *, subject to constraints* $V(H) \geq V^0$, $h_1 \leq d_1$, $h_2 \leq d_2,..., h_{n-1} \leq d_{n-1}$ *for line with simple structure and rational weights* $b_1,..., b_{n-1}$, V^0 *and* λ_i, μ_i, u_i, $i = 1,..., m$ *is NP-hard.*

Proof. Consider a special case of the problem with $m=2(n-1)$, where each chain consists of two sequential machines indexed i, $i+1$, $i=1 \bmod 2$. Let $j=(i+1)/2$ be the index of the buffer between machines i, $i+1$. Assume also that $\lambda_i = 2u_i$, $\mu_i = 4u_i$ for all $i = 1,..., m$, $d_1 = d_2 = ... = d_{n-1} = 1$ and $u_i = u_{i+1}$ for all $i=1 \bmod 2$. By means of formulas from (Dolgui, 1993) in case $h_j=1$ we obtain: $V'_j = 8u_{2j}/13$, where V'_j is the throughput of chain with buffer j; and in case $h_j=0$ we have: $V'_j = u_{2j}/2$. The throughput of the whole system is $V(H) = V'_1 + V'_2 + ... + V'_{n-1}$ and thus all necessary system parameters are computable in polynomial time in length of problem input encoding.

To prove the *NP*-hardness of the described problem, we reduce the partition problem (Garey and Johnson, 1979) to the described problem. The optimization version of partition problem is *NP*-hard and it can be formulated as follows:

$$Min \sum_{j=1}^{N} a_j y_j ,$$

$$\sum_{j=1}^{N} a_j y_j \geq \frac{1}{2} \sum_{j=1}^{N} a_j ,$$

where $a_1, a_2, ..., a_N$ are integer and the variables $y_1, y_2, ..., y_N$ belong to $\{0,1\}$. The required reduction is obtained by setting $n=N+1$, $u_{2j}= u_{2j+1} =a_j 26/ 3$, $b_j=a_j$, $j=1,2,\ldots,n-1$ and $V^0=\frac{29}{6}\sum_{j=1}^{N} a_j$. Indeed, in case $h_j=1$ we have $V'_j= a_j \cdot 16/3$ and otherwise $V'_j= a_j \cdot 13/3$. Therefore the total throughput is

$$V(H)= \sum_{j=1}^{n-1} V'_j = \sum_{j=1}^{n-1} a_j h_j + \frac{13}{3}\sum_{j=1}^{n-1} a_j = \sum_{j=1}^{n-1} a_j h_j + V^0 - \frac{1}{2}\sum_{j=1}^{n-1} a_j .$$

The objective functions are identical. □

Acknowledgments

The research is supported by INTAS (projects 00-217 and 03-51-5501). The authors thank Chris Yukna for checking the English of this paper.

References

Altiok T. (1996) *Performance analysis of manufacturing systems*, Springer, New York.

Buzacott J. A. and Hanifin L.E. (1978) Models of automatic transfer lines with inventory banks: a review and comparison, *AIIE Transactions*, **10** (2), 197-207.

Buzacott J. A. and Shanthikumar J. G. (1993) *Stochastic models of manufacturing systems*, Prentice Hall, New Jersey.

Coillard P. and Proth J.M. (1984) Effet des stocks tampons dans une fabrication en ligne, *Revue belge de Statistique, d'Informatique et de Recherche Opérationnelle*, **24** (2), 3-27.

Dallery Y. and Gershwin S.B. (1992) Manufacturing flow line systems: a review of models and analytical results, *Queuing Systems Theory and Applications*, Special Issue on Queuing Model of Manufacturing Systems, **12**, 3-94.

Dallery Y., David R. and Xie X. (1989) Approximate analysis of transfer lines with unreliable machines and finite buffers, *IEEE Transactions on Automatic Control*, **34**, 943-953.

De Koster M.B.M. (1987) Estimation of line efficiency by aggregation, *International Journal of Production Research*, **25**, 615-626.

Dolgui A. (1993) *Analyse de performances d'un atelier de production discontinue: méthode et logiciel*, Research Report INRIA 1949, 44 pages.

Dolgui A., Eremeev A., Kolokolov A., and Sigaev V. (2002) A genetic algorithm for the allocation of buffer storage capacities in a production line with unreliable machines, *Journal of Mathematical Modelling and Algorithms*, **1** (2), 89-104.

Dolgui A.B. and Svirin Y.P. (1995) Models of evaluation of probabilistic productivity of automated technological complexes, *Vesti Akademii Navuk Belarusi*: *phisika-technichnie navuki*, n°1, 59-67 (in Russian).

Droste S., Jansen T. and Wegener I. (2006) Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, **39** (4), 525-544.

Dubois D. and Forestier J.P. (1982) Productive et en-cours moyens d'un ensemble de deux machines séparées par une zone de stockage, *RAIRO Automatique*, **16** (2), 105-132.

Garey M.R. and Johnson D.S. (1979) *Computers and Intractability. A Guide to the theory of NP-completeness,* W.H. Freeman and Company, San Francisco.

Gershwin S. B. (1987) An efficient decomposition method for the approximate evaluation of tandem queues with finite storage space and blocking, *Operations Research*, **35** (2), 291-305.

Gershwin S.B. (1994) *Manufacturing systems engineering*, Prentice Hall, New Jersey

Gershwin S.B. and Schor J.E. (2000) Efficient Algorithms for Buffer Space Allocation, *Annals of Operations Research*, **93**, 117-144.

Glasserman P. and Yao D.D. (1996) Structured buffer-allocation problems. *Journal of Discrete Event Dynamic Systems*, **6**, 9-42.

Glover F. and Laguna M. (1997). *Tabu search*. Kluwer Academic Publishers.

Helber S. (2001) Cash-flow-oriented buffer allocation in stochastic flow lines, *International Journal of Production Research*, **39**, 3061-3083.

Inman R. R. (1999). Empirical evaluation of exponential and independence assumptions in queueing model of manufacturing systems, *Production and Operations Management*, **8** (4), 409-432.

Jafari M.A. and Shanthikumar J.G. (1989) Determination of optimal buffer storage capacities and optimal allocation in multistage automatic transfer lines, *IIE Transactions*, **21** (2), 130-135.

Levin A.A. and Pasjko N.I. (1969) Calculating the output of transfer lines, *Stanki i Instrument*, **8**, 8-10 (in Russian).

Li J. (2005). Overlapping decomposition: a system-theoretic method for modeling and analysis of complex manufacturing systems, *IEEE Transactions on Automation Science and Engineering*, **2** (1), 40-53.

Motwani R. and Raghavan P. (1995). *Randomized algorithms*. Cambridge University Press.

Papadopoulos H.T. and Heavey C. (1996). Queueing theory in manufacturing systems analysis and design: A classification of models for production and transfer lines, *European Journal of Operational Research*, **92**, 1-27.

Patchong A., Lemoine T. and Kern G. (2003). Improving car body production at PSA Peugeot Citroen, *Interfaces,* **33** (1), 36-49.

Reeves C.R. (1993). *Modern heuristic techniques for combinatorial problems*, John Wiley & Sons, New York, NY.

Shi L. and Men S. (2003). Optimal buffer allocation in production lines, *IIE Transactions*, **35**, 1-10.

Sörensen K. and Janssens G.K. (2004). A Petri net model of a continuous flow transfer line with unreliable machines. *European Journal of Operational Research*, **152**, 248-262.

Spinellis D. and Papadopoulos C. (2000). Stochastic algorithms for buffer allocation in reliable production lines, *Mathematical Problems in Engineering*, **5**, 441-458.

Tempelmeier H. (2003). Practical considerations in the optimization of flow production systems, *International Journal of Production Research*, **41** (1), 149-170.

Terracol C. and David R. (1987). An aggregation method for performance valuation of transfer lines with unreliable machines and finite buffers, *Proceedings of the IEEE International Conference on Robotics and Automation*, 1333–1338.

Vouros G.A. and Papadopoulos H.T. (1998). Buffer allocation in unreliable production lines using a knowledge based system, *Computers and Operations Research*, **25** (12), 1055-1067.

Yamashita H. and Altiok T. (1998). Buffer capacity allocation for a desired throughput in production lines, *IIE Transactions*, **30**, 883-891.

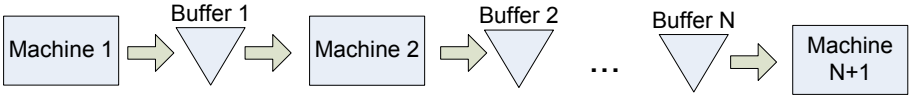

Figure 1. Tandem production line

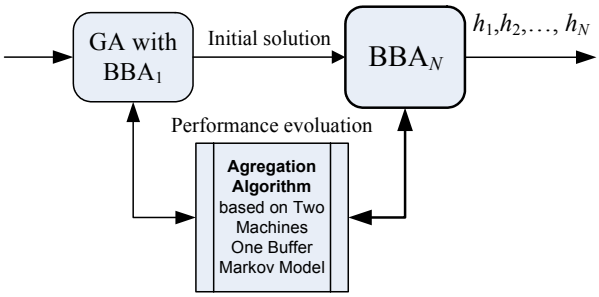

Figure 2. Optimization approach

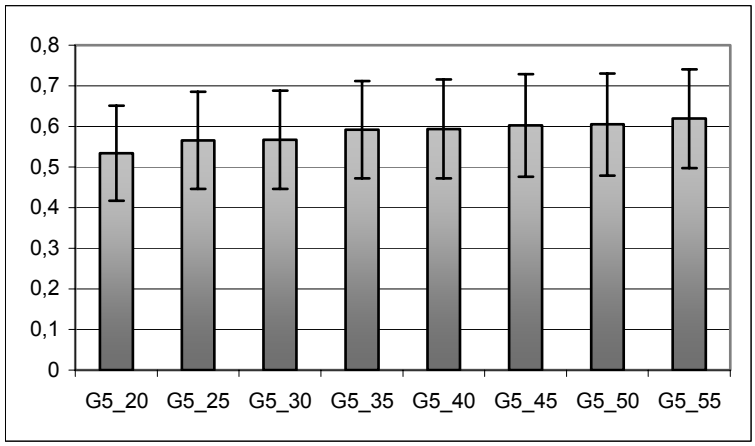

Figure 3. Average relative speed-up of HBBA vs. BBA for the series **G5_*m***.

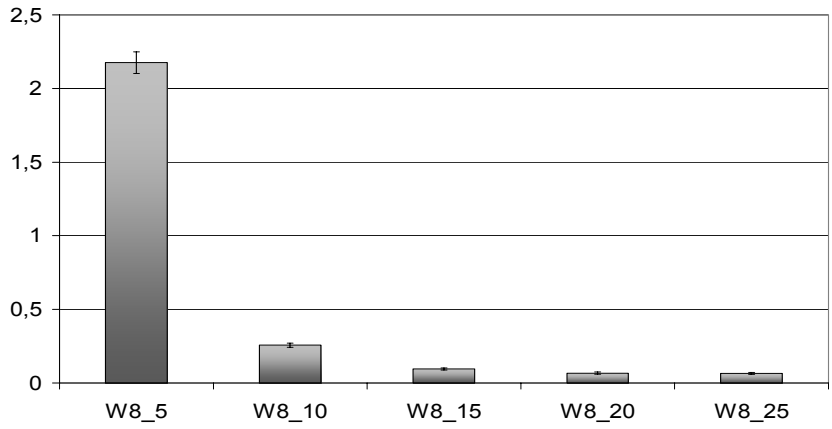

Figure 4. Average relative speed-up factor of HBBA vs. BBA for the series **W8_*m***.

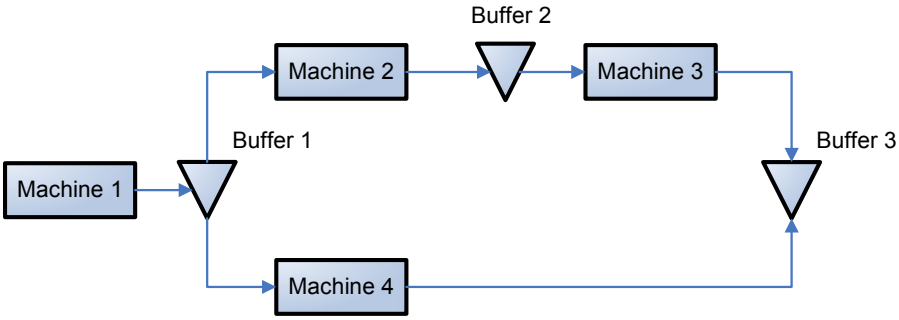

Figure 5. The structure of the line for the counterexample

N	Series		
5	**G5_20**	**G5_25**	**G5_30**
5	**G5_35**	**G5_40**	**G5_45**
5	**G5_50**	**G5_55**	
6	**G6_20**	**G6_25**	**G6_30**
6	**G6_35**	**G6_40**	
7	**G7_15**	**G7_20**	**G7_25**
8	**G8_10**	**G8_15**	**G8_20**
8	**W8_5**	**W8_10**	**W8_15**
8	**W8_20**	**W8_25**	

Table 1. Randomly generated series

Series	t_{CEM}	Running time of HBBA			t_{GA}	*Sol*
		$t^{\min}_{HBBA}$	t_{HBBA}	$t^{\max}_{HBBA}$		
G5_20	48.1	0.45	1.33	5.15	0.56	4.8E+04
G5_25	146.7	0.44	2.25	8.44	0.59	1.0E+05
G5_30	365.1	0.46	3.77	17.69	0.62	2.0E+05
G5_35	789.2	0.50	6.54	36.65	0.69	3.7E+05
G5_40	1538.6	0.52	10.98	69.59	0.72	6.5E+05
G5_45	2772.7	0.53	17.78	125.22	0.76	1.1E+06
G5_50	4695.5	0.56	28.00	213.66	0.79	1.7E+06
G5_55	7562.2	0.57	43.05	356.53	0.82	2.7E+06
G6_20	1146.8	0.77	19.01	145.31	0.75	9.5E+05
G6_25	4374.7	0.79	47.97	355.79	0.80	2.5E+06
G6_30	13062.7	0.79	105.79	699.66	0.85	5.5E+06
G6_35	32939.3	0.89	214.42	1565.49	0.95	1.1E+07
G6_40	73394.9	0.88	376.53	2989.73	1.00	2.0E+07
G7_15	244.2	0.80	13.92	93.10	0.89	5.8E+05
G7_20	1372.0	0.93	53.88	325.69	1.01	2.4E+06
G7_25	5233.8	0.94	159.75	983.51	1.06	7.2E+06
G8_10	2464.5	0.89	23.77	208.47	1.03	8.7E+05
G8_15	63163.3	0.96	190.85	1782.55	1.14	7.3E+06
G8_20	630920.5	1.12	929.16	8786.98	1.30	3.6E+07

Table 2. Running times of CEM and HBBA

Problem	H	φ	$\tau_{BBA}{}^{i}$
vp6_3	(0,2,5,3)	0.3359	0.000
vp6_4	(0,0,3,1)	0.3602	0.010
vp6_5	(0,1,3,1)	0.3782	0.010
vp6_6	(0,1,3,2)	0.3946	0.020
vp6_7	(0,1,4,2)	0.4094	0.020
vp6_8	(0,2,4,2)	0.4215	0.030
vp6_9	(0,2,5,2)	0.4334	0.040
vp6_10	(5,3,2,0)	0.4427	0.050
vp7_3	(2,1,0,0)	0.4515	0.010
vp7_4	(2,1,1,0)	0.4605	0.020
vp7_5	(3,1,1,0)	0.3359	0.030
vp7_6	(3,2,1,0)	0.3602	0.050
vp7_7	(4,2,1,0)	0.3782	0.071
vp7_8	(4,3,1,0)	0.3946	0.111
vp7_9	(5,3,1,0)	0.4094	0.130
vp7_10	(5,3,2,0)	0.4215	0.170

Table 3. Running times of the Pure BBA and the solutions obtained

Machine 1		Machine 2		Machine 3		Machine 4		Buffer 3
A	**D**	**A**	**D**	**A**	**D**	**A**	**D**	**A**
0	1	1	2	2	4	–	–	4
1	2	2	4	4	6	–	–	6
2	3	–	–	–	–	3	4	4
3	4	4	6	6	8	–	–	8
4	5	–	–	–	–	5	6	6

Table 4. First case timetable

Machine 1		Machine 2		Buffer 2		Machine 3		Machine 4		Buffer 3
A	**D**	**A**	**D**	**A**	**D**	**A**	**D**	**A**	**D**	**A**
0	1	1	2	2	2	2	4	–	–	4
1	2	2	3	3	4	4	6	–	–	6
2	3	3	4	4	6	6	8	–	–	8
3	4	4	6	6	8	8	10	–	–	10
4	5	–	–	–	–	–	–	5	6	6

Table 5. Second case timetable