INT. J. PROD. RES., 2004, VOL. 42, NO. 6, 1189–1205

Taylor & Francis
Taylor & Francis Group

Minimization of open orders: application of a re-distribution coordination policy

ALEXANDRE LINHARES†*

This work considers an NP-Hard scheduling problem that is fundamental to the production planning of flexible machines, to cutting pattern industries and also to the design of VLSI circuits. A new asynchronous collective search model is proposed, exploring the search space in a manner that concentrates effort onto those areas of higher perceived potential. This is done with the use of a coordination policy which enables the processes with greatest performance to act as 'attractors' to those processes trapped in areas of worse perceived potential. Numerical results are obtained for problems of realistic industrial size, and those results are compared to previously published optimal solutions. This comparison demonstrates the effectiveness of the method, as in 276 problems out of a set of 280 we are able to match previously reported optimal results.

1. Minimization of open orders problem

Let us consider the following production planning problem: a flexible machine capable of producing a set of distinct items must be scheduled with every item of each specific type being produced in a complete production run. Customer orders are composed of a combination of item types, and each order remains open from the moment its first item type is produced until the very last item type of the order is produced (at which point the order is closed and may be shipped to the customer). It is of interest to plan production as to have the number of open unfinished orders as small as possible. As has been pointed out (Yanasse 1997), unfinished orders can yield additional logistical overhead and associated costs, such as higher inventory costs, higher lead times (due to delayed deliveries and delayed payments), split deliveries (and associated higher transportation costs), etc. It is thus of interest to have the number of open client orders at any point in time as low as possible. Let us refer to this problem as the minimization of open orders problem, hereafter MOOP.

This problem also arises in other industrial settings: consider a cutting setting where large boards are cut into smaller pieces. Each of the pieces is placed in a corresponding stack which will hold all items of the same type. A stack remains open – occupying space around the cutting saw – from the moment the first piece of its type is cut until the moment the last piece of its type is cut, at which point the stack may be removed. Obviously, there is a maximum capacity of stacks that may be placed around the cutting saw, and in case the number of stacks exceeds capacity, some of them will have to be removed and later brought back for additional

Revision received July 2003.

*Brazilian School of Business and Public Administration, EBAPE/FGV, Praia de Botafogo 190/509, Rio de Janeiro 22257–970, Brazil. E-mail: linhares@fgv.br

†National Institute of Space Research, LAC/INPE, Av Astronautas 1758, S.J.Campos, SP12227–010, Brazil.

International Journal of Production Research ISSN 0020–7543 print/ISSN 1366–588X online
http://www.tandf.co.uk/journals
DOI: 10.1080/00207540310001622421

processing – yielding a problem of minimizing the number of stack switches, which is known as MTSP – see for instance, Tang and Denardo (1988), Bard (1988), and also Shirazi and Frizelle (2001). In order to plan production to remain below system capacity, it is desired to minimize the maximum number of open stacks, a problem termed minimization of open stacks problem, or MOSP (Yanasse 1997, Linhares and Yanasse 2002a).

These goals lead us to the computational problems of obtaining a proper permutation of the production runs (in the MOOP) and of the cutting patterns (in the MOSP). Computational complexity studies have found the MOOP to be an NP-Hard problem, and furthermore, that there can be no absolute approximation algorithm for its solution, unless the unlikely scenario of P = NP turns out to be true (Linhares and Yanasse 2002a). It has also been pointed out that these problems additionally appear in VLSI design settings (Linhares and Yanasse 2002a).

Let us formulate the problem precisely: There are J distinct item types, each requiring a corresponding production run; and I customer orders. The relationship between these objects can be defined by a $I \times J$ binary matrix $P = \{p_{ij}\}$, with $p_{ij} = 1$ iff item j is contained in order i, and $p_{ij} = 0$ otherwise. An open orders versus production runs matrix $Q^{\pi} = \{q^{\pi}_{ij}\}$ is defined by:

$$q^{\pi}_{it} = \begin{cases} 1, & \textit{if } \exists x, \exists y | \pi(x) \leq t \leq \pi(y), \textit{ and } p_{ix} = p_{iy} = 1 \\ 0, & \textit{otherwise} \end{cases} \tag{1}$$

where π denotes a permutation of the (1,2, . . ., J) numbers, and defines a sequence of production runs, such that $\pi(i)$ is the order (instant) in which the ith item type is produced. Note that matrix Q^{π} holds the consecutive-ones property for columns (Golumbic 1980) under permutation π: in each row, any zero between two ones will be changed to one. This is called a fill-in. Since we are interested in minimizing the number of simultaneous open orders, I define the following cost functional:

$$Z^{\pi}_{MOOP}(P) = \max_{j \in \{1, \ldots, J\}} \sum_{i=1}^{I} q^{\pi}_{ij} \tag{2}$$

and define the MOOP as the problem of $\min^{\pi \in \Gamma} Z^{\pi}_{MOOP}(P)$, where Γ is the set of all possible one-to-one mappings π:$\{1, 2, \ldots, J\} \overset{\pi \in \Gamma}{\rightarrow} \{1, 2, \ldots, J\}$.

In this paper, a new collective search model for this production planning problem is proposed. The next section will look at previous collective search models in the literature.

2. Collective local search models

Solution methods for the MOOP, in its numerous facets (as the MOSP, or as a problem of VLSI design), have been proposed by several authors: computational experiments have been reported for the MOSP in (Fagiolli and Bentivoglio 1998, Fink and Voss 1999). The best reported results for VLSI circuits have appeared in (Linhares *et al.* 1999). Overall, these experiments have applied exact branch and bound solution schemes for problems of moderate size and modern local search heuristics for industrial-sized problems; they have not, however, considered more advanced, collective search methods.

In Linhares *et al.* (1999), a fast statistical mechanics method is restarted multiple times in order to obtain multiple high quality solutions – and eventually exceptionally high quality ones (which, though conjectured to be optimal, could not be proved

to be so). This type of approach may be seen as having multiple asynchronous independent processes with no communication between them. However, additional information may be used to dynamically redirect search processes, and it may be advantageous to employ some form of communication between processes in order to improve performance of the system as a whole.

Toulouse *et al*. (1996) classified three useful (tough rather informal) questions in the design of such collective methods: First, when should information be shared? And what information should be shared? Finally, where should information be shared? The question of when to share information concerns which events should trigger the intercommunication processes. The question of *what* information should be shared concerns how much the search patterns should alter, and also the level of detail of information exchanged between processes. Finally, the question of where information should be shared concerns which specific individual processes should be communicating (and which should be receiving information).

Let us look at a specific example. In another paper they presented a set of 'cooperating processes' that, over some particular moments in course of the execution, exchange their 'configurations', that is best solutions found (see Toulouse *et al*. 1998). Thus, new processes (those which were not obtaining the best solutions) are effectively drawn to areas that present high potential. In this model exact copies of previously explored solutions are frequently reinstantiated and, to a certain extent, this configures an undesirable duplication of search effort. Another characteristic of this model is reduced diversity: if search processes are unconstrained to wander over the search space, a certain distribution will naturally emerge; if, however, processes are regularly placed back to previously searched points, the resulting distribution may have processes clustered in smaller areas of the search space. It is no wonder that the authors come to the conclusion that 'diversity of the configuration of [interacting, solution-exchanging,] populations is [. . .] much lower than the same population of programs that do not interact with each other in any manner'. (p2380) The risks associated with lower population diversity, obviously, concern inability to find high quality solutions residing far from the 'solution cluster', and the eventual, undesirable, duplication of search effort.

They deduce that it is advantageous to exchange this sort of information, as 'most empirical studies conclude that a population of search programs exchanging gathered data obtains better performance compared to the same population of non-interacting programs' (Toulouse *et al*. 1998, p 2379): this is one of the premises of the model proposed here, which generalizes that presented by Toulouse *et al*. (1998).

The sharing of information between search processes is the key of our proposal. We suggest a redistribution coordination policy, that is, a control mechanism for guiding search processes based on information acquired by other processes, the objective of which is to redirect those processes that are seemingly trapped in poor regions (and unable to find high-quality solutions) to more promising areas of the search space. The policy selects potentially good areas by taking advantage of the information obtained by the remaining search processes. As detailed below, at $J/3$ local search steps (where J is the number of item types) the very worst process is re-directed to a new area by moving it (along a minimum path) towards the best solution found by a successful process while still preserving a specific distance from that best solution. In this manner it is expected that areas with greater perceived potential will have a concentrated search effort; however, the undesirable effect of having the very same solutions revisited is not obtained (as is the case for the

algorithm presented in Toulouse *et al.* 1998). The model for each individual search process will be detailed below.

3. Model implementation

Let us begin by presenting a step-by-step pseudo-code of the redistribution policy, and then discussing each of its steps.

Pseudo-code of redistribution policy

```
begin
  while (There exists at least one process P running) do
  begin
  Process[P].executestep; {intensive search or sampling}
  If (Process[P] found a new best solution) then
Re-Sort Position of Process P;
  If (counter reaches time for redistribution policy) then
  Begin
      Select Worst Positioned Process as Psource;
      Randomly select one of the three best positioned (and still
      running) processes as PTarget;
      If there is no better positioned running process, halt
      redistribution;
      Otherwise Redistribute Psource to a region close to
      PTarget;
  end;
    update variable P for the next running process;
  end;
end;
```

This is the main loop of the algorithm. The first step is for a process to perform the routine named executestep.

3.1. *Executestep procedure*

This routine implements each individual search process, the architecture of which is that used in Linhares *et al.* (1999): a solution (production sequence) is represented by a permutation $\pi = (\pi_1, \pi_2, \pi_3, \ldots, \pi_n)$, new solutions are generated by the 2-exchange operator, which switches two positions p_1 and p_2 in a permutation, that is, it transforms permutation $(\pi_1, \ldots, \pi p_1, \ldots, \pi p_2, \ldots, \pi_n)$ to $(\pi_1, \ldots, \pi p_2, \ldots, \pi p_1, \ldots, \pi_n)$. In the neighbourhood implied by the 2-exchange operator, the mechanics for redistributing psource to a region close to ptarget may be obtained by halting the distance algorithm in figure 1 at the particular desired point. The linear time algorithm presented in figure 1 computes the minimum 2-exchange distance between two permutations π and i, where $i = (1, 2, \ldots, N)$ is the identity permutation. It should be obvious that there is no loss of generality for using i, as the distance from π to any permutation σ can be obtained simply by recognizing the relation that $D(\pi,\sigma) = D(\sigma^{-1}\pi, i)$ (see for instance Caprara 1999 or Linhares 2001). To use this distance function to make psource approach ptarget to a distance of at most Δ, it resorts to halting the process described in figure 1 whenever $d \leq \Delta$. For a proof of correctness (and of optimality) of this algorithm we refer the reader to (Linhares 2001).

```
d = 0;
for i= 1 to n do
      if π_i≠σ_i then begin
      2-EXCHANGE(i, σ^-1(π_i) );
      d = d +1;
      end;
return d;
```

Figure 1. A critical element of the proposed method: the 2-exchange distance metric.

The objective function and our particular evaluation have been detailed in Linhares and Torreão (1998) and in Linhares *et al.* (1999). Also following Linhares *et al.* (1999), the search processes alternate between two search phases: one of intensive search, when the search process is guided towards a local minimum solution and another termed sampling, when the search is directed towards finding a new region. Let us consider each one in turn.

The intensive search phase is characterized by three aspects: (1) it only accepts improving solutions, since the foremost goal is to obtain a locally-minimum solution; (2) at each step, it samples 4% of the neighbourhood (hence it is a function of the size of the problem) and selects the best sample, as long as it is improving. We have found in experiments that the quality of the results obtained when less than 4% of the neighbourhood is sampled decreases considerably, and a complete (or significantly larger) sample places great demand on the execution time, as the cost evaluation of a neighbouring solution is not a straightforward $O(n)$ process – see Fink and Voss (1999). This phase stops when $J/2$ subsequent proposed moves are non-improving and are rejected, where J is the number of item types (corresponding to patterns in MOSP, gates in LGAP or columns in the column permutation).

On the other side, in the sampling phase, non-improving moves may be accepted if they respect a certain cost boundary. This boundary in our illustration is set as follows: from the current solution, 10 neighbouring solutions are evaluated and the second best cost is selected. Since there is a great probability that the last phase stopped at a local minimum solution, these 10 neighbouring solutions usually have higher cost than the current solution. Thus, the second best cost among these is a practical value (i.e. known to exist in that context) that is not significantly worse than the current cost, at least in relation to the (random) sample. Thus, there will be a number of solutions that are acceptable from that particular solution (that is, they have better cost than the selected boundary). The process accepts these solutions, preserving large part of the optimization performed since the start of the execution, while still moving to another region. This phase stops whenever 50 solution movements are attempted. Each search process is set to stop whenever 10 phases are executed without obtaining any improvement on the cost of the best solution found. A pseudo-code of each individual process and of the sampling and initialization phases is available at (Linhares *et al.* 1999).

3.2. *Redistribution coordination policy*

A redistribution coordination policy, intended to concentrate the search over areas of higher perceived quality, is active. Processes are ordered by their performance (measured by the quality of the best solutions found). Hence, at each $J/3$ steps (measured by the process 1 counter), the current solution of the worst process is selected to act as the source position, and it is made to approach a target position. The target position is selected randomly from the best solutions found by the three most successful processes. A shortest path is computed between the source and the target, and the new current solution of the (previously) worst process was set to approach the target, while remaining at seven movements of distance from it. This specific parameter was selected (after a number of trials) because it was the smallest number that, in practice, enabled processes to approach targets while still maintaining a distance in order to prevent duplication of search effort. As I argue in the following section, with this parameter setting, processes do approach each other closely, but at no point does the distance drop to absolute zero (i.e. the wasted effort of passing through the very same solutions). After the redistribution coordination policy is executed, processes are re-ordered based on the best cost achieved by each one.

3.3. *Population size*

The number of concurrent search processes selected for execution in the single-processor case was 20. (Unfortunately, the current implementation only simulates parallelism: distinct processes alternate their execution at each local search step [i.e. visited solution]. It is a desirable research goal to develop a truly parallel algorithm and have it execute asynchronously in a machine with multiple processors, but we must leave such a goal for future research.) Obviously, it is desirable to employ a large number of processes, as a greater part of the search space may be explored at once. Also, a larger number of potential interactions between processes will directly influence the collective behaviour of the model. On the other hand, it is costly to add more processes as they affect the performance of the system, which in our single-processor case is seen to gradually degrade to a point of exhaustion whenever more than 25 processes are executed. Thus, given these tradeoffs, we have selected 20 concurrent processes as the best cost-benefit parameter for our (simulated parallel) tests. The whole algorithm stops whenever the last process stops executing.

4. Search dynamics and performance

My concern in this section is with the collective dynamics of the processes. Let us start by studying the distribution of the search processes. A natural question here concerns whether or not diversity will be lost with the use of the coordination policy, and whether there will be undesirable duplication of search effort (whenever two processes visit the very same solution). I have conducted an experiment to understand the natural distribution of the search processes without interactions between them, that is without using the redistribution coordination policy. The experiments suggest that for this problem processes do not cluster and tend naturally to maintain a widespread distribution over the search space. This fact will be clearly seen in the following figures. Given 20 independent search processes, one has calculated the minimum distance from each process to the corresponding closest process (we refer the reader interested in these distance metrics to Linhares and Yanasse 2002b). Each sampling step corresponds to five local search steps (measured by

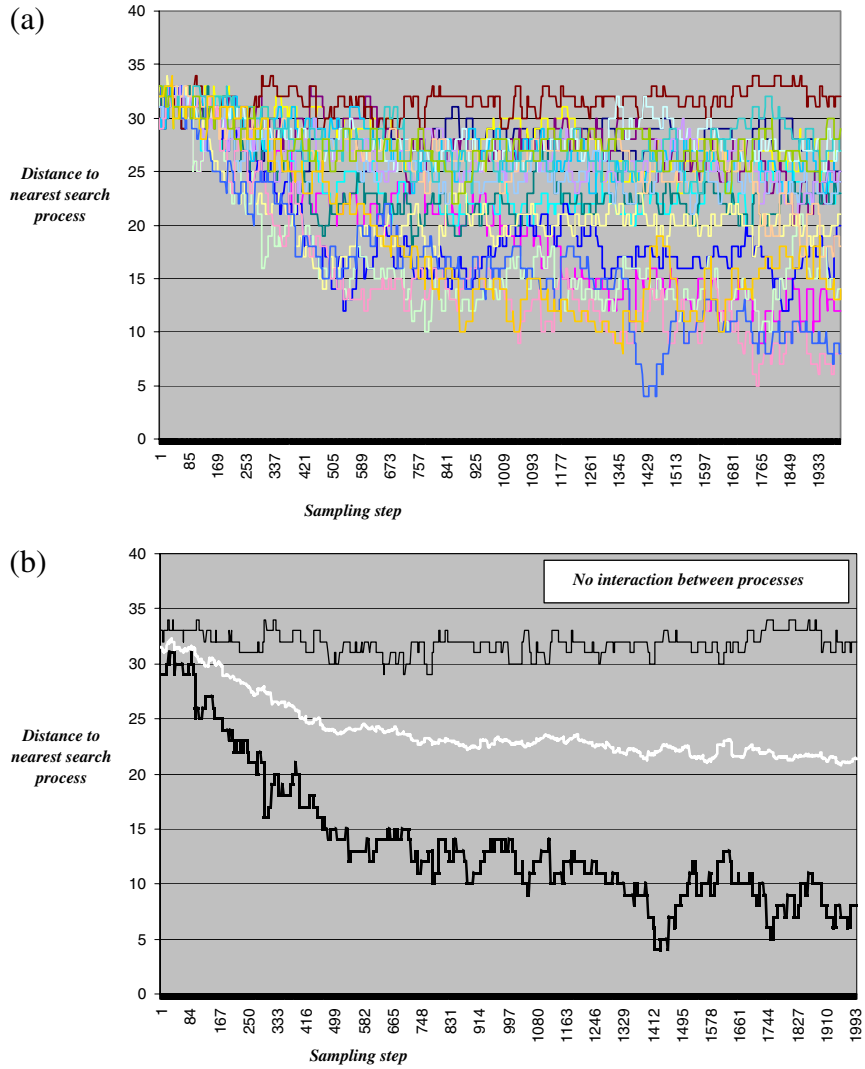

Figure 2. Distances maintained by independent processes: (a) distance of all processes to their nearest counterparts; (b) Corresponding minimum closest-pair distance, average closest-pair, and maximum closest-pair.

the counter of process 1). The plot in figure 2(a) presents the evolution of these measures over the whole execution of the system. Since the display is cluttered, I will be working with displays such as that of figure 2(b), which focuses only on the corresponding minimum, maximum and average values corresponding to those given by all processes.

These results have been obtained in the first problem (#0) with 40 item types and 50 orders from Faggiolli and Bentivoglio (1998) – that is, test problem (40,50,0). These results are representative of our larger sample, which is the full set of 300 problems used in Faggiolli and Bentivoglio (1998). The problems were generated randomly and are subdivided in 30 sets of 10 problems. The 30 sets arise from combinations of the number of item types $J \in \{10, 15, 20, 25, 30, 40\}$ and the

number of orders $I \in \{10, 20, 30, 40, 50\}$, with 10 problems being generated for each combination of these values in the following manner: the 'composition of every pattern was obtained by randomly extracting four piece types [orders] from a uniform distribution over the interval [1, I]' (Faggiolli and Bentivoglio, 1998, p571); there is also a final check for completeness, to ensure that all item types need to be produced (Faggioli and Bentivoglio 1998). This check alters the average density of the matrices, such that the density is not fixed at $4/I$ (the reader should consult Fagiolli and Bentivoglio [1998] for additional details).

Notice that the average values tend to display a gradual descent over time (starting at around 32 movements, gradually moving towards 25 and, finally, 20 at the end of the run) but at no point in time do processes move extremely close to each other, as, for instance, having less than four movements of distance. More importantly, at no point do two distinct processes consider the very same solutions. Over the course of the whole execution, the minimum distance between closest pairs is four moves but the maximum distance between closest pairs reaches more than 30 movements. And if we consider other, non-closest, pairs of processes, distances increase dramatically. For example, in figure 3, in which the corresponding graph for the farthest pairs of processes is presented, it is clear that processes are widely distributed throughout the search space, as the average is higher than 35 moves, while the maximum attainable in this landscape is 39 movements. Similar results have been reproduced in all the other test problems considered. Given a specific implementation, it remains as an empirical question whether processes will cluster and thus have such a wasteful duplication of search effort. Our data demonstrate that this is not the case for this particular application, but it still remains to be seen whether this will hold for other problem domains.

Let us contrast that to the data obtained whenever the redistribution coordination policy is active. Figure 4 is the counterpart to figures 2(b) and 3, presenting the maximum, average, and minimum values of the distance between each process and its nearest counterpart (in figure 4[a]) or its farthest counterpart (in figure 4[b]). In a direct comparison between figures 2(b) and 4(a), we may note some facts: (1) as

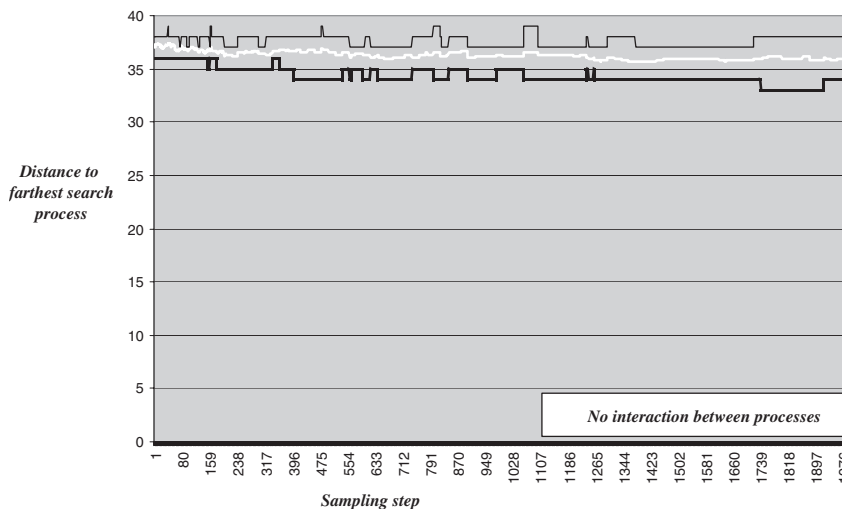

Figure 3. Distance to the farthest process when processes are independent.

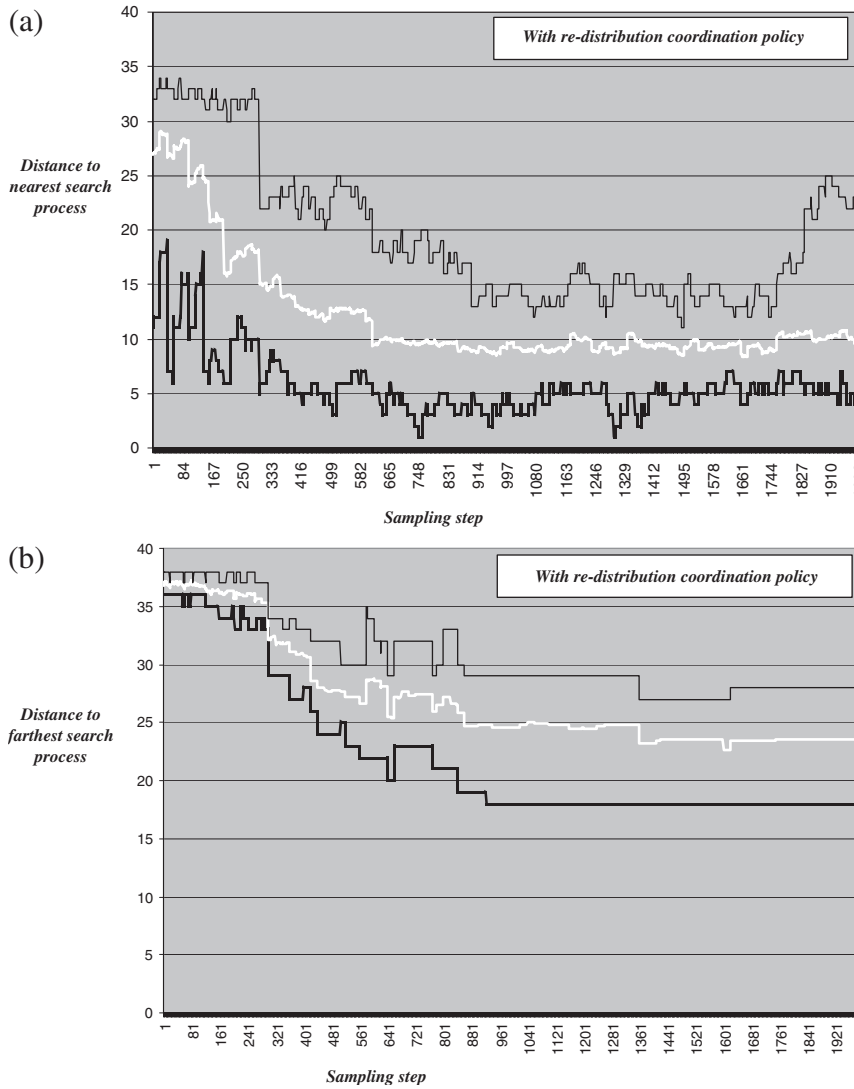

Figure 4. Distances with interaction between processes (re-distribution coordination policy is active): (a) distance between closest pairs of processes; (b) distance to farthest pairs of processes.

expected, the maximum closest-pair distance drops considerably, and this fact supports the intuition that processes which were in separate (distant) regions are often the processes that have found solutions of worst quality and thus, are the ones selected for re-distribution; and (2) the corresponding average and minimum distances also drop considerably; in this case, the average values, which were between 20 and 25 movements, drop to the neighbourhood of 10 movements, while the minimum distances drop from the interval of [5,15] to the tighter interval of [4,7] (and going as low as two movements twice). As we have mentioned, seven movements of distance were selected for the implementation of the coordination policy precisely because that was the smallest value that enabled processes to approach

targets while subsequently maintaining a small distance (in order to prevent duplication of search effort).

What does this mean for the diversity of the search? Note once again that this is the average of the distance between closest pairs of processes. The average distance to other, non-closest, pairs of processes is significantly larger. For instance, as can be seen in figure 4 (b), which displays the (minimum, average and maximum) distances to the farthest processes (with interaction), the average distance to the farthest process lies, after the initial convergence, in the interval of 20 to 25 movements. Without interaction, as seen in figure 3, the maximum distance lies close to the maximum possible distance, which is, in this case, 39 movements. Even when a redistribution coordination policy is active, the average distance between closest pairs lies around 10 movements and the average distance between farthest pairs lies around 25 movements, with the distance between other pairs of processes generally residing in the interval of [5,30] movements. In summary, the high distribution of the search is preserved as all 20 interdependent processes lie on the interval of 10 (average distance to the closest) to 25 movements (average to farthest) in a space with a diameter of 39 movements.

4.1. *Interaction effects on the cost of solutions*

Let us now consider how the application of the redistribution coordination policy affects the cost of the solutions obtained. In figure 5 the evolution of the maximum, minimum and average cost searched by the multiple processes is shown for the problem instance dealt with in figures 2–4. Part (a) deals with the independent processes (no active coordination policy – which corresponds to independent runs of the Linhares *et al*. [1999] algorithm), while part (b) deals with interacting processes (the re-distribution coordination policy is active). The cost effects can be readily perceived in this graph just as the distance effects were perceivable in the preceding figures. All cost values (the maximum, average, and minimum) obtained by the processes drop dramatically when the policy is active and there is interaction between processes. The maximum values drop considerably more than the others, as expected, for the obvious reason that the worst processes are being moved to new areas (which are close to high quality solutions). As can be seen on the graph, this movement also affects to a large extent the average values and the minimum values converge faster and more robustly.

Note that the object of study is the effect of the alteration of search area in the cost obtained by a search process. The redistribution coordination policy brings processes to a new area (which is closer to one of the best solutions found, while still preserving a certain distance from those solutions). Processes that have found high quality solutions exercise influence over less successful processes, 'attracting' the latter to potentially more promising areas.

The results presented in figure 5 are also representative of the other test problems of the set. In figure 6 a considerably larger set of problems is considered: the 50 largest problems (with 40 item types each) considered by Faggioli and Bentivoglio (1998). Each dot displays the results obtained by the system when the policy is active divided by the results obtained by a non-interacting system (i.e. the policy is inactive). In this way we can perceive the gains of using the coordination policy. Each dot corresponds to (the executions with and without the policy on) a single problem. The axii display the relative number of visited solutions versus the relative cost obtained

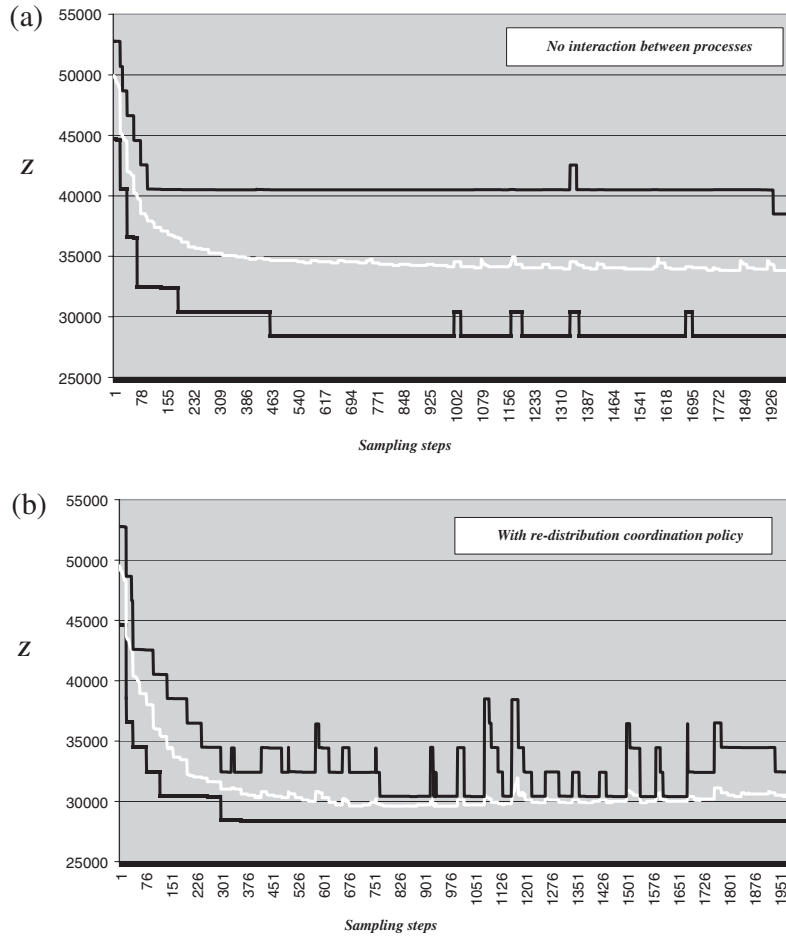

Figure 5. Interaction cost effects.

(by averaging the cost of the best processes over a run of the system). Let us start with the relative number of solutions visited.

In the graph it is clearly shown that the relative number of solutions has a much higher variance than the relative cost of the solutions obtained. Variance for the relative number of solutions is, in fact, 0.008894, and the average of the (number of visited solutions) data is 1.000824, indicating that, on average, the number of solutions visited when the policy is active remains remarkably close to the corresponding number for a system without such policy. The maximum value is 1.25674 while the minimum is 0.755882. This is easy to explain since there is no relation between the coordination policies and the termination criterion used in our system; thus the number of solutions should not increase or decrease with the use of the re-distribution coordination policy.

At the other axis, the relative cost of solutions (computed as the average cost of the best processes in tests with redistribution coordination policy divided by the average costs obtained by the best processes of a system without coordination policies) found by the system drops, on average, to 0.942654 of that obtained without the use of the policy. Variance is considerably smaller than that of the number of

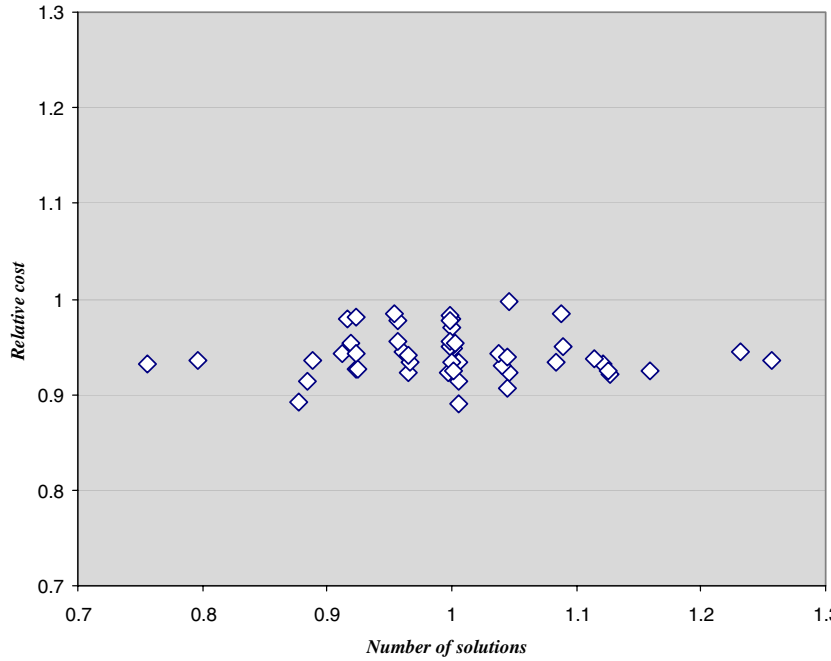

Figure 6. Average cost of the best solutions visited and number of solutions relative to those obtained by independent processes without the use of re-distribution coordination policy.

solutions, at 0.000574. The maximum value is 0.997732 (which corresponds to a marginal gain), and the minimum value is 0.889787 (which is a significant gain). Note that these results are in relation to those results obtained by the non-interacting system which corresponds to that proposed in Linhares *et al.* (1999), and is by itself capable of finding high-quality solutions. What this figure displays, basically, is the improvement on the quality of solutions found without an increase on the number of solutions visited, a sign of the increased intelligence of the search generated by the re-distribution coordination policy. This may be true for instances exhibiting a 'big-valley' structure where there should be incentive to concentrate the search effort to those areas of higher perceived quality. However, there are still instances for which the improvement found is negligible.

The collective model is more robust than its counterpart without coordination policy. For instance, consider the circuit problems dealt with in Linhares *et al.* (1999), which is the basis for implementation of each individual search process: in problem **wli** (from Linhares *et al.* 1999) microcanonical optimization was able to find the optimum solution in only 36.3% of 1000 trials – a disappointing performance, given the relatively small size of the circuit. The collective algorithm presented here has found the optimum in 100% of 100 trials, taking less than a second for each execution. In the considerably larger circuit **w3**, the collective model is able to find the best known solution of 18 tracks in 50% of 100 trials (averaging 18.67 tracks per run, in 52 seconds of execution). If the re-distribution coordination policy is not active, however, the optimum solution is obtained in only 34 (out of 100) trials.

In the following section our numerical results are compared with the exact and approximate results obtained by Faggioli and Bentivoglio (1998).

4.2. *Performance evaluation and comparison with optimal results*

Three major solution methods are presented in the paper by Faggioli and Bentivoglio (1998). The first method is an implicit enumeration (a branch and bound approach) that finds optimum solutions (and supposedly takes a large amount of time). The second method is a tabu search procedure based on an optimized move selection process that, at each step, reinserts a pattern in the best possible position. The third method is a 'generalized local search' procedure that basically works by employing multiple applications of a simplified tabu search that only accepts improving moves; which was found by Faggioli and Bentivoglio (1998) to be quite competitive with tabu search. These latter heuristics, the paper concludes, are better alternatives relative to a number of previous heuristics such as those presented by Yuen (1991, 1995) and Yuen and Richardson (1995).

Table 1 contrasts our results with those obtained by Faggioli and Bentivoglio (1998), presenting, on each row, the data for each set of ten problems. The columns

J	I	OPT	Collective	Tabu Search	GLS
10	10	5.5	5.5	5.5	5.5
	20	6.2	6.2	6.2	6.2
	30	6.1	6.1	6.1	**6.2**
	40	7.7	7.7	7.7	7.7
	50	8.2	8.2	8.2	8.2
15	10	6.6	6.6	6.6	6.6
	20	7.2	7.2	7.2	**7.5**
	30	7.4	**7.3 (?)**	7.4	**7.6**
	40	7.3	**7.2 (?)**	7.3	**7.4**
	50	7.6	**7.4 (?)**	7.6	7.6
20	10	7.5	7.5	**7.7**	7.5
	20	8.5	8.5	**8.7**	**8.6**
	30	8.8	**9**	**9.2**	**8.9**
	40	8.6	8.6	8.6	**8.7**
	50	7.9	7.9	**8**	**8.2**
25	10	8	8	8	8
	20	9.8	9.8	9.8	**9.9**
	30	10.5	**10.6**	**10.7**	**10.6**
	40	10.4	10.4	**10.7**	**10.6**
	50	10	10	**10.1**	**10.2**
30	10	7.8	7.8	7.8	7.8
	20	11.1	**11.2**	**11.2**	**11.2**
	30	12.2	12.2	**12.6**	12.2
	40	12.1	12.1	**12.6**	**12.4**
	50	11.2	11.2	**12**	**11.8**
40	10	8.4	8.4	8.4	8.4
	20	13	13	**13.1**	**13.1**
	30	14.5	14.5	**14.7**	**14.6**
	40	15	15	**15.3**	**15.3**
	50	14.6	14.6	**15.3**	**14.9**

Table 1. Comparison with reported optimal results. Entries in bold deviate from the reported optima of Faggioli and Bentivoglio (1998).

display, in the following order: the corresponding number of item types (i.e. matrix columns), the number of customer orders (i.e. matrix rows), the average objective value of the optimum result obtained on the ten problems, the average objective value obtained by our collective method, for their tabu search and for their 'generalized local search'. The entries emphasized in bold are deviations from the reported supposedly optimum solution values. Observe that some of the entries in the OPT column have values actually higher than obtained; we believe that this is due to typos in the (Faggioli and Bentivoglio, 1998) table (note that the OPT values for these problems match the reported values for the tabu search method. I believe that this might be the reason for such typos. Anyway, I presuppose that the optimum results hold for the remaining sets of problems) and I consider their remaining results as correct (and hence optimum). I have been able to employ the exact algorithm presented in Yanasse (1997) and solve the set of 15 patterns and 30 items to optimality, obtaining 7.3 open orders (stacks) on average. (This has been done with help from Marcelo S. Limeira's exact branch and bound algorithm. Unfortunately, it has not been possible to obtain the corresponding results for the remaining sets of problems due to the exponential demands in computational time.)

The method was implemented on an object-oriented Pascal compiler and the results clearly show that it is capable of finding optimum solutions for a large set of problems. In the 30 sets of problems our method was able to produce optimum solutions for at least 25 full sets. This means that all the ten problems in each one of the sets have been solved to optimality. Observe also that I am not including the sets where my results are actually better than the reported optima, as I have no indication that these solutions are indeed optimum (I do, however, based on extensive further experimentation, conjecture that such is the case). Thus, should I decide to ignore these sets of problems, then I may argue that there was a deviation of only four additional open orders (stacks) over the whole (remaining) set of 280 problems! These results testify for the high quality of the solutions obtained by the collective model. I do, however, make no claims whatsoever in terms of a time comparison with the heuristic methods used in Faggioli and Bentivoglio (1998), for a number of reasons. First, because there is not enough information (besides a time-growth curve) presented in Faggioli and Bentivoglio (1998) to make such a comparison meaningful. Furthermore, the hardware that was utilized there dramatically differs from mine (I do believe that, under idealized conditions, their single-process methods might be faster than our collective proposal). In figure 7 the computational times growth as a function of the number of item types is displayed. Those numbers were obtained on a Pentium IV CPU 1.5 Ghz, using the sets from Faggioli and Bentivoglio (1998) with the number of customers orders fixed at 50. The numbers are averages for the 10 problems contained in each set. It is clear that the proposed method does not require significantly large amounts of time. Moreover, our collective method is naturally prone to improvement in a (potentially much faster) parallel implementation. I believe that a more meaningful comparison to other approaches should be based on the number of visited solutions – a measure that is independent of hardware and of implementation and can be used even for comparisons between serial and parallel methods. Thus, in table 2 I present the average number of solutions visited for each set of problems. It is easy to see that this number grows with the number of item types but not with the number of customers orders. Interestingly, in four sets of problems, the average number of visited solutions of the problems with the smallest number of customers' orders was actually larger than the corresponding number for

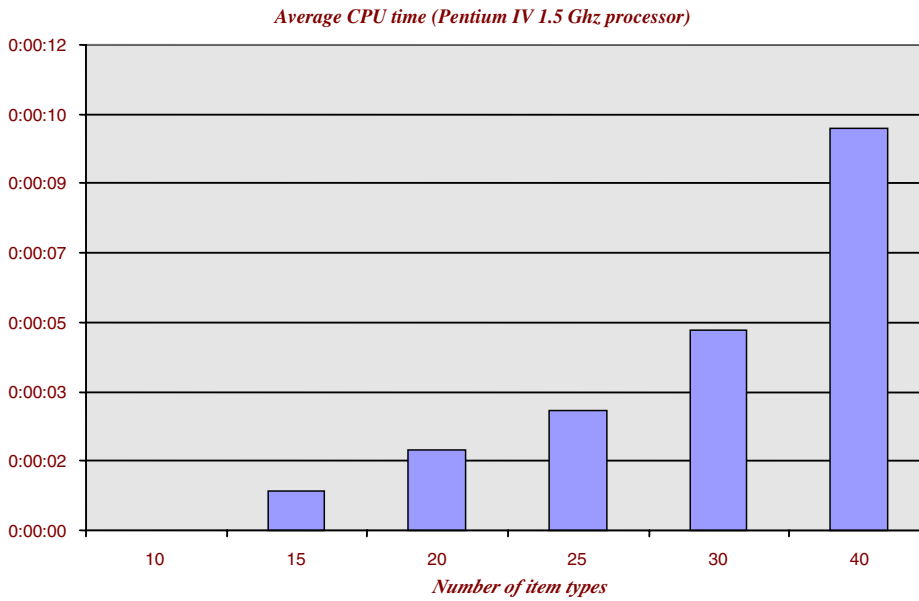

Figure 7. Execution time growth.

		Item types					
		10	15	20	25	30	40
Orders	10	1598.0	2940.6	4815.3	5718.0	7353.6	10020.1
	20	1546.1	2902.5	4833.4	5809.9	7262.7	10527.4
	30	1543.0	2923.3	4606.2	5746.1	7383.9	10509.2
	40	1540.2	2931.7	4669.3	6040.6	7614.6	10201.2
	50	1513.9	2930.2	4636.5	5893.4	7588.6	10005.3

Table 2. Number of solutions visited by all processes (averages taken over each set of problems).

the problems with the largest number of orders. Notice that the stop criterion does not bear any relation to the number of orders.

These values are the totals for all the 20 search processes and thus, each process, including the one(s) that found the best solution(s), have had, on average, 1/20 of this number of visited solutions. This brings us to a figure of about 500 visited solutions per process on the largest problems, which contain 40!/2 distinct solutions. Thus, each process has only a 'glimpse' of an incredibly vast search space, and yet the system still manages to find, collectively, optimum solutions. As mentioned, this implementation only simulates parallelism (and synchronous parallelism at that) but the real gains should come from a truly parallel, asynchronous, concurrent execution of the model in a (set of) machine(s) with multiple processors. It remains to be seen whether, in this environment, the processes could visit concurrently such a small number of solutions and the whole system could conclude the search in a fraction of the time it takes to conclude the search in a serial (but simulated parallel) execution, but we must leave this is as a topic for further research.

5. Conclusion

In a previous paper, I presented new coordination policies and distance metrics for advanced collective search; we have also introduced a framework that challenges the frequent view in the literature that 'search intensity and search diversity are mutually exclusive'; that 'these desirable characteristics can not be achieved under perfect simultaneous co-existence' – see Linhares and Yanasse (2002b). In this paper I present new coordination policies for extending the model of Linhares *et al.* (1999), that enable the system to 'attract solutions to areas of high perceived quality', while not taking them to previously visited solutions. The intended philosophy is to concentrate the collective search to areas of high perceived potential without the undesirable duplication of search effort which arises when distinct processes explore the same solution.

The reader should note that this approach generalizes the collective model of the 'cooperating tabu search' presented in Toulouse *et al.* (1998). In that reference, cooperation amounts to a 'swapping of configurations' between search processes, that is, a search process transfers a complete solution to another process. This case is more general, because processes are brought closer to the best configurations found, avoiding the wasted duplication of search effort. Toulouse *et al.* (1998) also remarked that cooperating (configuration-swapping) processes exhibit less diversity than regular independent processes, and this also happens here (see Linhares 2001). The method has been capable of matching previously reported optimum solutions to a set of 276 out of 280 problems in a reasonably short amount of time: less than 10 s per instance. It seems that the use of redistribution coordination policies enables a more effective and intelligent search process without increasing the number of visited solutions.

Note finally that this is just an initial study and many important algorithmic design possibilities remain open. Even if processes do not tend to cluster around small regions in our case, this cannot be generalized to other sequencing or scheduling problems. Once again, the whole architecture of the proposed collective model is intended for parallel implementations, which can be executed asynchronously over multiple processors. Interesting questions for future research in this line include the interprocess communication that is necessary for implementing these promising coordination policies.

References

Bard, J. F., 1988, A heuristic for minimizing the number of tool switches on a flexible machine. *IIE Transactions*, **20**, 382–391.

Caprara, A., 1999, Sorting permutations by reversal and Eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, **12**, 91–110.

Fagiolli, E. and Bentivoglio, C. A., 1998, Heuristic and exact methods for the cutting sequencing problem. *European Journal of Operational Research*, **110**, 564–575.

Fink, A. and Voss, S., 1999, Applications of modern heuristic search techniques to pattern sequencing problems. *Computers & Operations Research*, **26**, 17–34.

Golumbic, M. C., 1980, *Algorithmic Graph Theory and Perfect Graphs* (London: Academic Press).

Linhares, A., 2001, Industrial pattern sequencing problems: some complexity results and new local search models, PhD thesis, Brazilian Institute of Space Research, São José dos Campos.

Linhares, A. and Torrão, J. R. A., 1998, Microcanonical optimisation applied to the travelling salesman problem. *International Journal of Modern Physics*, **C9**, 133–146.

Linhares, A. and Yanasse, H. H., 2002a, Connections between cutting pattern sequencing, VLSI layouts, and flexible machines. *Computers & Operations Research*, **29**(12), 1759–1772.

Linhares, A. and Yanasse, H. H., 2002b, Local search intensity versus local search diversity: a false tradeoff? Manuscript submitted for publication.

Linhares, A., Yanasse, H. H. and Torreão, J. R. A., 1999, Linear gate assignment: a fast statistical mechanics approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **18**(12), 1750–1758.

Shirazi, R. and Frizelle, G. D. M., 2001, Minimizing the number of tool switches on a flexible machine: an empirical study. *International Journal of Production Research*, **39**, 3547–3560.

Tang, C. S. and Denardo, E. V., 1988, Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches. *Operations Research*, **36**, 767–777.

Toulouse, M., Crainic, T. G. and Gendreau, M., 1996, Communication issues in designing cooperative multi-thread parallel searches. In Osman, I., and Kelly, J. P. *Meta-heuristics: Theory and Practice* (Norwell, Ma: Kluwer Academic Publishers), pp. 501–522.

Toulouse, M., Crainic, T. G., Sansó, B. and Thulasiraman, K., 1998, Self-organization in cooperative tabu search algorithms. Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics, San Diego, CA, pp. 2397–2385.

Yanasse, H. H., 1997, On a pattern sequencing problem to minimize the maximum number of open stacks. *European Journal of Operational Research*, **100**, 454–463.

Yuen, B. J., 1991, Heuristics for sequencing cutting patterns. *European Journal of Operational Research*, **55**, 183–190.

Yuen, B. J., 1995, Improved heuristics for sequencing cutting patterns. *European journal of Operational Research*, **87**, 57–64.

Yuen, B. J. and Richardson, K. V., 1995, Establishing the optimality of sequencing heuristics for cutting stock problems. *European Journal of Operational Research*, **84**, 590–598.