PERGAMON

Computers & Operations Research 29 (2002) 1641–1659

computers & operations research

www.elsevier.com/locate/dsw

A new evolutionary approach to cutting stock problems with and without contiguity

Ko-Hsin Liang[a,*], Xin Yao[b], Charles Newton[a], David Hoffman[a]

[a]*School of Computer Science, University College, The University of New South Wales, Australian Defence Force Academy, Canberra, ACT 2600, Australia*

[b]*School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, UK*

Received 1 August 1998; received in revised form 1 July 1999

Abstract

Evolutionary algorithms (EAs) have been applied to many optimization problems successfully in recent years. The genetic algorithm (GAs) and evolutionary programming (EP) are two different types of EAs. GAs use crossover as the primary search operator and mutation as a background operator, while EP uses mutation as the primary search operator and does not employ any crossover. This paper proposes a novel EP algorithm for cutting stock problems with and without contiguity. Two new mutation operators are proposed. Experimental studies have been carried out to examine the effectiveness of the EP algorithm. They show that EP can provide a simple yet more effective alternative to GAs in solving cutting stock problems with and without contiguity. The solutions found by EP are significantly better (in most cases) than or comparable to those found by GAs.

Scope and purpose

The one-dimensional cutting stock problem (CSP) is one of the classical combinatorial optimization problems. While most previous work only considered minimizing trim loss, this paper considers CSPs with two objectives. One is the minimization of trim loss (i.e., wastage). The other is the minimization of the number of stocks with wastage, or the number of partially finished items (pattern sequencing or contiguity problem). Although some traditional OR techniques (e.g., programming based approaches) can find the global optimum for small CSPs, they are impractical to find the exact global optimum for large problems due to combinatorial explosion. Heuristic techniques (such as various hill-climbing algorithms) need to be used for large CSPs. One of the heuristic algorithms which have been applied to CSPs recently with success is the genetic algorithm (GA). This paper proposes a much simpler evolutionary algorithm than the GA, based on evolutionary programming (EP). The EP algorithm has been shown to perform significantly better than the GA for most benchmark problems we used and to be comparable to the GA for other problems.

Keywords: Evolutionary algorithm; Cutting stock problems; Combinatorial optimization

* Corresponding author. Fax: + 886–7–5850347.

E-mail address: liangk@cna.edu.tw (K.-H. Liang).

0305-0548/02/$ - see front matter
PII: S0305-0548(01)00039-9

1. Introduction

The cutting stock problem (CSP) is an important class of combinatorial optimization problems [1–3]. The traditional goals of CSPs are to minimize the trim loss and/or the cost value. Dyckhoff and Finke [2] categorized seven different objectives according to the characteristics of CSP. Most CSP solution methods are designed for specified objective functions.

In this paper, two classes of one-dimensional CSPs are considered. The common objective for both CSPs is the trim loss minimization, also called wastage minimization. Each CSP has a second objective. For the first class of CSPs, the second objective is to minimize the number of used stocks. This type of CSP with two objectives has been tackled by using some heuristic techniques [4–6]. For the second class of CSPs, the second objective is to minimize the number of partially finished (open) items. This is known as the cutting pattern sequencing problem [7] or CSP with contiguity [6].

Traditional OR approaches to one-dimensional CSPs [3] can be divided into two categories, heuristics and linear programming (LP)-based methods [1]. Golden [8], Hinxman [9] and Haessler and Sweeney [10] have written excellent surveys on the various solution approaches. LP-based methods are designed solely to minimize trim loss or cost value. They have difficulties in dealing with non-linear problems which are most common in the real world. As a result, heuristic methods have become increasingly popular in solving real-world CSPs. However, the effectiveness and efficiency of a heuristic algorithm depend heavily on the heuristics used. Finding good heuristics is often as difficult as solving the problem itself. In addition, most good heuristics are highly problem-dependent.

In the past decade, evolutionary algorithms (EAs) [11] have become promising methods in solving various optimization problems. The evolutionary approach is particularly good at dealing with complex and nonlinear problems and finding a near-optimal solution within a reasonable amount of time. Genetic algorithms (GAs) [12] and evolutionary programming (EP) [13] are two major classes of EAs. While GAs have been applied to the cutting and packing problems by many researchers [6,14–18], EP's application in combinatorial optimization is relatively limited [19,20].

Crossover is the primary search operator in GAs. It combines sound building blocks from different parents and passes them to their offspring. However, the effectiveness of crossover as the primary search operator is problem-dependent. It works very well for some problems, but not for others.

EP is simpler than GAs since it only uses mutation. It has been applied to a number of numerical and combinatorial problems with success [13]. It has been shown to be more efficient than GAs for some numerical optimization problems [21,22]. In this paper, we propose an EP algorithm with only a swap mutation operator which outperforms GAs for one-dimensional CSPs with and without contiguity.

The mutation in our EP is designed using the concept of the distance between a parent and its offspring. The distance is defined as the difference between two permutation lists with the same set of elements. For a given list, a two point swap is defined as the exchange of an arbitrary pair of elements in the list. The distance between the original list and the resulting list is thus defined as 1. Such a concept of distance between permutation lists was first introduced and applied to the traveling salesman problem [23,24].

The rest of this paper is organized as follows. Section 2 describes CSP with and without contiguity. In both cases, two objectives have to be considered and minimized. Section 3 introduces new mutation operators used in our EP algorithm and gives implementation details. Section 4 presents the experimental results of our study and compares them with GA's results. It is shown that EP has always performed at least as well as, and most times significantly better than, the GA for all ten benchmark problems using less computational time. Finally, Section 5 concludes the paper with a brief summary of the paper and some remarks.

2. The cutting stock problem

The aim of CSP is to cut an object made of material that can be a reel of wire, paper, or a piece of wood, etc., to satisfy customers' demands. The material is referred to as the stock of "(large) objects" and the list of demands as so-called "(small) items" [1]. If there is more than one stock length to be cut to fulfill the requests, the problem is called a multiple stock length CSP. In this paper, we consider both CSPs with single stock lengths and CSPs with multiple stock lengths.

2.1. CSP without contiguity

In addition to the issue of single versus multiple stock lengths, contiguity is another important issue in the CSP. When a list of items are cut we need to consider storage space requirements where partially finished items are placed until the requested number of items is completed or the packaging size is met. This is referred to as the *contiguity* issue in CSP. It is a more realistic model of real world situations, especially when the problem size becomes large.

Most traditional LP-based and heuristic methods only consider CSP without contiguity. In this case, two objectives of the one-dimensional CSP are to minimize the wastage and the number of used stock with wastage. Mathematically, we would like to minimize the following cost function:

$$C = \sum_{j=1}^{m} f(w_j, V_j), \tag{1}$$

where $f(w_j, V_j)$ is a function of w_j and V_j

$$w_j = L_j - \sum_{i=1}^{n} x_{ij} l_i, \quad j = 1, \ldots, m, \tag{2}$$

$$V_j = \begin{cases} 1 & \text{if } w_j > 0, \\ 0 & \text{otherwise,} \end{cases} \quad j = 1, \ldots, m \tag{3}$$

subject to

$$\sum_{j=1}^{m} x_{ij} = N_i, \quad i = 1, \ldots, n, \tag{4}$$

where n is the number of different requested items, m the total number of stocks cut, w_j the wastage of the jth stock, V_j the jth stock with wastage, L_j the stock length of the jth stock, x_{ij} the number of

the ith requested item in the jth stock, l_i the length of the ith requested item, and N_i the total number of the ith requested item.

2.2. CSP with contiguity

CSPs with contiguity are often more difficult to solve. A heuristic approach to this problem is to determine the cutting pattern of each stock first, and then sequence the cutting order of each stock. CSPs with contiguity also have two objectives, i.e., minimizing wastage and contiguity. The cost function for CSPs with contiguity can be formulated as

$$C = \sum_{j=1}^{m} f(w_j, o_j) \tag{5}$$

where $f(w_j, o_j)$ is a function of w_j and o_j

$$o_j = \sum_{i=1}^{n} y_i, \tag{6}$$

$$y_i = \begin{cases} 0 & \text{if } \sum_{k=1}^{j} x_{ik} = 0 \text{ or } \sum_{k=1}^{j} x_{ik} = N_i, \\ 1 & \text{otherwise,} \end{cases} \tag{7}$$

where o_j is the number of partially finished (open) orders up to the jth cutting stock, and y_i is the open status of the ith requested item. According to Eq. (7), an item is open (i.e., $y_i = 1$) if we have started cutting this item from stocks but have not completed the whole order (i.e., N_i). Constraints Eqs. (2) and (4) given in Section 2.1 also apply in the case of CSPs with contiguity. Other definitions and notations have been given in Section 2.1 and will not be repeated here.

2.3. Evolutionary approaches to CSPs

The only evolutionary approach which has been used to solve CSPs with and without contiguity is based on GAs [6]. Two representations have been used in the GA approach, i.e., order- and group-based representations [6]. The group-based representation uses groups of items as genes, and the number of genes is equal to m, the total number of stocks used. The order-based representation uses an ordered list to represent all the items to be cut. A decoder, therefore, is needed to organize the cutting points and the stock length of each point in the list. It has been shown that the group-based GA is better than the order-based GA for CSPs without contiguity, and comparable to the order-based GA for CSPs with contiguity [6]. One of the major problems with the order-based GA is that crossover is unable to exploit the ordering information in the chromosome representation. In the group-based GA, the ordering information is encapsulated within groups (genes) and less susceptible to the destruction of crossover.

The implementation of the crossover operator in GAs is normally a difficult task. The crossover has to maintain the feasibility of offspring, otherwise some kind of penalty or repair methods must be used in order to evolve a feasible solution. The order-based GA uses uniform order-based crossover (UOB) [25] that applies a template of randomly generated binary bits to exchange some

items and maintain relative order information from both parents. The mutation operator uses both swap and remove and reinsert (RAR) [6]. The group-based GA uses grouping crossover, where a segment of one parent is inserted into another parent to generate an offspring [18]. Such crossover may not be as straightforward as it first appears because it has to avoid duplicated items being copied into the offspring. The mutation operator selects a number of genes and rebuilds new genes from the selected genes, which is a much more time-consuming task than a simple swap mutation. The RAR mutation is also used in the group-based GA.

Hinterding and Khan [6] have observed that the crossover in the order-based GA could degrade the performance of GA. The crossover operator could not exploit the ordering information and sometimes it even destroyed the ordering information [6]. In this paper, we propose a new evolutionary approach to CSPs with and without contiguity based on an EP algorithm. The EP algorithm is much simpler and less time-consuming than the GA, and only uses swap mutations. No crossover is adopted. The order-based representation will be used in our EP algorithm, which turns out to be much better than the group-based GA.

3. The evolutionary programming approach

To solve the CSP using EP, three major modifications to the classical EP (CEP) [22] are made. They are (1) problem representation, (2) mutation operator, and (3) fitness evaluation function.

3.1. Problem representation

In most numerical optimization instances, CEP uses a pair of real-valued vectors as an individual to represent a candidate solution. One vector contains the object variables and the other the adaptive parameters. For the CSP considered in this paper, only one integer vector, $\vec{x}$, is used. It is an ordered list of all requested items. No self-adaptation is used in our experiments. In other words, there are no additional parameters to evolve or adapt in our EP.

Fig. 1 shows an example of order-based representation and the mapping from the representation to a solution. Given stocks of length 12 (single stock length), and the requests of 2 items of length 3, 2 items of length 4, 1 item of length 5, and 3 items of length 6. The first row in Fig. 1 shows an order list of all items requested, which is a "chromosome". The second row shows how this "chromosome" will be decoded into a solution by including cut points. The method to decide each cut point is intuitive. A cut is made before the accumulated item length matches any stock length, or the accumulated item length exceeds the available stock length. The third row shows the wastage for each stock.

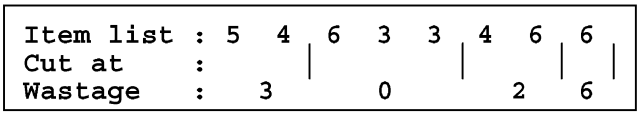

Fig. 1. Eight requested items are represented as an ordered list. The stock length is 12. Four stocks are used here. The total wastage is 11.

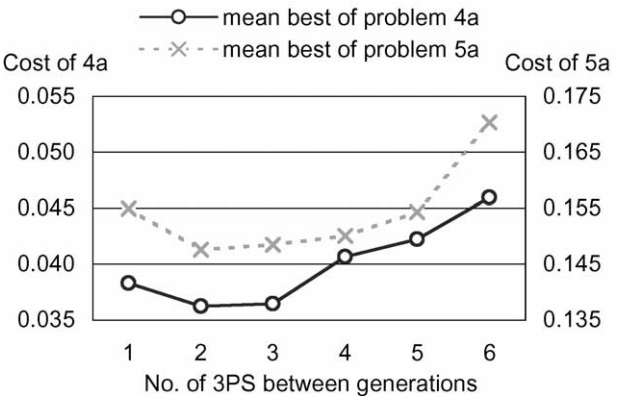

Fig. 2. The impact of different numbers of 3PS in one mutation on the evolutionary results for problems 4a and 5a [6]. Each result is averaged over 50 independent runs, where "mean best" is the mean best values found at the 2000th generation. The vertical axis represents the cost value and the horizontal axis indicates the number of 3PS used in one mutation.

3.2. Swap mutation

Unlike the CEP for numerical optimization [22], the item itself in a "chromosome" cannot be changed. For example, we cannot change the first item 5 in figs. 1–4 because 5 indicates the user request. What we should change or mutate is the order. There are a number of ways one can mutate the order. We propose a simple mutation operator based on a three point swap (3PS) in this paper. The idea of this mutation comes from the concept of distance between two permutation lists [23]. Given two permutations with n elements

$$X = (x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$$

and

$$Y = (y_1, \ldots, y_i, \ldots, y_j, \ldots, y_n), \quad 1 < i < j < n,$$

the minimum distance between them can be defined as $D(X, Y) = 1$ if $x_i = y_j$, $x_j = y_i$, and $x_k = y_k$ for all other k's. In other words, the distance between two permutation lists is determined by the minimum number of pair-wise swaps to go from one list to the other.

In theory, a global optimal solution can be reached through a sequence of swaps with distance 1 from any initial population of permutation lists. However, this could be a very slow process and might get stuck in a local optimum for a long time. Large search step sizes through multiple swaps may be beneficial [23]. In this paper, we introduce 3PS as a basic operator which swaps 3 items in a list. 3PS swaps the first item with the second one, and then swaps the new first one with the third one. For each 3PS, the resultant permutation list moves distance 2 away from its original list. If we apply 3PS multiple times within one mutation, a permutation list may move even further away from its original list. This may accelerate the convergence towards the global optimum if the original list is far away from the global optimum. It may also slow down the convergence if the global optimum is already very close to the original list. A balance has to be maintained between

the large-step exploration by applying several 3PS's in one mutation and the small-step exploitation by applying only one 3PS.

Fig. 2 shows two examples of how different numbers of 3PS in one mutation can affect the final result of the EP algorithm. The test problems used in these examples are problems 4a and 5a from [6], which are single stock length CSPs with 60 and 126 requested items, respectively. These two problems are harder to solve than others given in [6].

Two observations can be made from the results in Fig. 2. First, as expected, the number of 3PS used in one mutation did influence the performance of the EP algorithm. Neither a large nor small number was good. Second, both 2 and 3 gave good performances, which indicated that there was a small range for a suitable number of 3PS. It was unnecessary to fine tune this number as long as it was within a certain range. Two 3PS were used in one mutation in our study.

One important issue related to the number of 3PS in one mutation is the selection of the 3 items for 3PS. In our algorithm, the first item x_{ij} is selected uniformly at random from an ordered list. The second and third items are both selected in two steps. First, a stock j is selected at random according to the following probability:

$$\Pr(j) = \frac{\sqrt{1/w_j}}{\sum_{j=1}^{m} \sqrt{1/w_j}}, \quad \forall w_j \neq 0, \tag{8}$$

where w_j is the wastage of the jth stock. Then an item is selected uniformly at random from the stock.

On the surface, Eq. (8) appears to be counter-intuitive as it biases towards stocks with less wastage. It would have made more sense if we had selected stocks with more wastage more frequently in order to minimize the total wastage. However, CSPs considered in this paper are more complex than just minimizing the total wastage. The other objective is to minimize the number of stocks with wastage. The bias towards stocks with less wastage in Eq. (8) would discourage the EP algorithm to evolve solutions with many stocks each of which has a small wastage. This technique enables us to optimize two objectives by the EP algorithm.

For CSPs with contiguity, another mutation operator, the stock remove and insert (SRI) operator, is needed to consider the contiguity. To reduce the number of partially finished items while minimizing the total wastage, the best way is to appropriately rearrange the cutting sequence. The SRI operator does exactly this. It does not change any stock wastage w_j in Eq. (5). SRI is implemented by the following steps:

1. Select an item uniformly at random from an ordered list.
2. Remove the stock that cuts the selected item.
3. Search through the list to find the stock that cuts an item of the same length.
4. Insert the removed stock right behind the first such found stock.

Similar to the design of 3PS-based mutation, we may have more than one SRI within a single mutation in order to enhance the exploration ability of the SRI mutation. In our algorithm, 4 SRI were used in each mutation.

3.3. Fitness functions

In order to compare the effectiveness of the EP approach with the previous GA approach, we use the same fitness function as that in Hinterding and Khan [6]. For CSPs without considering contiguity, the cost to be minimized is defined as follows:

$$\text{Cost} = \frac{1}{m+1}\left(\sum_{j=1}^{m} \sqrt{\frac{w_j}{L_j}} + \sum_{j=1}^{m} \frac{V_j}{m}\right), \tag{9}$$

where the notations are the same as defined in Section 2.1. The cost function has two terms. The first term emphasizes minimizing the wastage and the second emphasizes minimizing the number of stocks with wastage.

For CSPs with contiguity the cost function is defined as

$$\text{Cost} = \frac{1}{m+10}\left(\sum_{j=1}^{m} \sqrt{\frac{w_j}{L_j}} + \frac{10}{m}\sum_{j=1}^{m} \left(\frac{o_j}{n}\right)^2\right), \tag{10}$$

where the notations are the same as defined in Section 2.2. This function uses the first term to minimize the wastage and the second one to minimize the number of open items.

3.4. The evolutionary programming algorithm

For CSPs without contiguity, our EP is implemented as follows:

1. Generate the initial population of μ individuals randomly, and set the generation counter $\kappa = 1$. Each individual is taken as an integer-valued vector, x_i, $\forall i \in \{1, \ldots, \mu\}$.
2. Calculate the cost value for each individual x_i, $\forall i \in \{1, \ldots, \mu\}$, in the population based on Eq. (9).
3. Each parent x_i, $i = 1, \ldots, \mu$, creates a single offspring x_i' using the 3PS mutation. 3PS are applied twice within the single mutation.
4. Calculate the cost value of each offspring x_i', $\forall i \in \{1, \ldots, \mu\}$.
5. Conduct pairwise comparisons over the union of parents x_i and offspring x_i', $\forall i \in \{1, \ldots, \mu\}$. For each individual, $q = 10$ opponents are chosen uniformly at random from all the parents and offspring. For each comparison, if the individual's cost is no greater than the opponent's, it receives a "win".
6. Select the μ individuals out of x_i and x_i', $\forall i \in \{1, \ldots, \mu\}$, that have the most wins to be parents of next generation.
7. Stop if the halting criterion is satisfied; otherwise, $\kappa = \kappa + 1$ and go to Step 3.

For CSPs with contiguity, two modifications are made to the above algorithm. First, the cost function is changed to Eq. (10). Second, in Step 3, either SRI or 3PS may be used in the mutation. The probability to execute the SRI mutation is 0.25. When executed, SRI will be applied 4 times in a single mutation.

4. Experimental studies

Although there is a rich literature on CSPs [1–10], few gave the actual data for their problems. Direct comparison with them is difficult if not impossible. In our experimental studies, 20 test problems are used. The detailed description of these problems are given in Appendix A. The 20 problems can be divided into two parts. The first part consists of ten benchmark problems obtained from Hinterding and Khan [6]. The total items requested range from 20 to 126. Problems 1–5 are multiple stock length CSPs and problems 1a to 5a are single stock length CSPs. Problems with the same index have the same number of requested items, e.g., problems 4 and 4a have the same number of requested items (60). The number of requested items increases as the problem index increases. Since these ten problems have been widely used by people in the evolutionary computation and operations research communities, they will help us in comparing the effectiveness of our new evolutionary approach, and facilitate the comparisons between the performances of the EP and GAs for the CSPs.

The second part of our test problems also consists of ten problems. They are much larger and more difficult than the previous ten. The number of requested items goes up to 600 for problems 10 and 10a. No evolutionary algorithms have ever been tested on such problems.

As mentioned in Section 1, traditional OR methods such as column generation [26] are designed mainly for single objective optimization. Once an optimal solution is found for a single objective, it is difficult to use the optimal cutting patterns for the second objective. To deal with two objectives, a heuristic approach [5] based on linear programming has been proposed. Unfortunately, the solutions found by this method normally do not satisfy equation 4 since the number of items generated can be greater than the demand. We have implemented a simple heuristic algorithm for the CSPs, i.e., the two-swap algorithm, which has been found to be effective for some combinatorial optimization problems. At each step (iteration) of this algorithm, two randomly selected items in an ordered list are exchanged. If the newly generated list has better fitness, it replaces the original one.

In the following experiments, the population size μ is 75, the same as that used by the GA [6]. The tournament size q is 10 in selection. Each problem is repeated for 50 independent runs. The experimental results on the first ten benchmark problems, i.e., problems 1–5 and 1a to 5a, are summarized in Tables 1 and 2, along with the previous results obtained by the GA [6]. Only the results of the group-based GA are shown, since it performed best among various GAs [6].

The results in Tables 1 and 2 show that our EP algorithm has performed at least as well as the GA for all ten benchmark problems. It performed significantly better for most problems. For CSPs without contiguity (Table 1), the EP algorithm has consistently found the global optimum in all 50 runs for six test problems, while the GA could only locate the optimum consistently for four problems. It is worth pointing out that it is impossible to obtain statistical significance (using the t-test) for eight out of ten problems. For the remaining two problems, the EP algorithm performed significantly better than the GA for one and similar to the GA for the other. For CSPs with contiguity (Table 2), the EP algorithm performed significantly better than the GA for six problems, and was comparable to the GA for the other four problems. It should be noted that our experiments were repeated for 50 independent runs while Hinterding and Khan [6] only repeated their experiments for 20 runs.

Table 3 summarizes the actual solutions found for the first ten problems by our EP algorithm. All results were averaged over 50 independent runs. It is quite clear from Table 3 that our EP

Table 1
Results of the group-based GA [6] and the order-based EP for CSPs *without* contiguity. "Req. item" is the total number of requested items. "Eval." indicates the number of cost function evaluations. "Gen" is the number of generations. "Found" indicates the generation number when the best solution was found

Prob	Req. item	GA, group based, no contiguity				EP, order based, no contiguity				EP-GA *t*-test
		Eval.	Mean	Std Dev	Found	Gen.	Mean	Std Dev	Found	
1	20	1184	0	0	407	20	0	0	4	0
2	50	1184	0	0	740	50	0	0	16	0
3	60	1184	0	0	407	100	0	0	8	0
4	60	2294	0.0005	0.0022	2294	300	0	0	70	− 1.0164
5	126	2294	0.0002	0.0007	2294	500	0	0	164	− 1.2778
1a	20	1184	0.0867	0	296	50	0.0867	0	26	0
2a	50	1184	0.0773	0.0018	1184	500	0.07688	0.00112	472	− 0.97432
3a	60	1184	0	0	407	300	0	0	112	0
4a	60	1184	0.0358	0	851	2000	0.03602	0.00155	1554	1.0003
5a	126	2294	0.1521	0.007	2294	2000	0.14318	0.00796	2000	− 4.6285[a]

[a]The *t*-value with 49 degrees of freedom is statistically significant at $\alpha = 0.05$.

Table 2
Results of the group-based GA [6] and the order-based EP for CSPs *with* contiguity. "Req. item" is the total number of requested items. "Eval." indicates the number of cost function evaluations. "Gen" is the number of generations. "Found" indicates the generation number when the best solution was found

Prob	Req. item	GA, group based, with contiguity				EP, order based, with contiguity				EP-GA *t*-test
		Eval.	Mean	Std Dev	Found	Gen.	Mean	Std Dev	Found	
1	20	2155	0.0140	0.0031	2155	500	0.00936	0.00117	475	− 6.5133[†]
2	50	5275	0.0204	0.0130	5275	1000	0.01408	0.00342	965	− 2.1461[†]
3	60	4235	0.0172	0.0041	4235	2000	0.01658	0.00311	2000	− 0.6087
4	60	5275	0.0390	0.0140	5275	2000	0.02613	0.00488	1825	− 4.0142[†]
5	126	10435	0.0111	0.0073	10435	2000	0.00728	0.00150	1995	− 2.3197[†]
1a	20	2155	0.0440	0	1739	500	0.0440	0	35	0
2a	50	5275	0.0788	0.0171	5275	1000	0.06836	0.00364	995	− 2.7067[†]
3a	60	4235	0.0613	0.0318	4235	2000	0.06268	0.01630	1985	0.1839
4a	60	5275	0.0665	0.0103	5275	2000	0.06572	0.00819	1820	− 0.3012
5a	126	10600	0.1531	0.0118	10600	2000	0.12506	0.00492	2000	− 10.276[a]

[a]The *t*-value with 49 degrees of freedom is statistically significant at $\alpha = 0.05$.

algorithm was able to cut stocks without any waste for six out of ten CSPs without contiguity and for five out of ten CSPs with contiguity. Table 3 also shows how well our EP algorithm did in optimizing two objectives simultaneously. For CSPs without contiguity, the number of stocks with wastage was either zero or a small number, as shown by the column in "Stocks *w*/wastage". The

Table 3
The mean best results for problems 1–5 and 1a to 5a over 50 independent runs for CSPs with and without contiguity, where "Stocks used" indicates the average number of stocks used, "Total wastage" indicates the average total wastage from all used stocks, "Stocks *w*/wastage" indicates the average number of stocks with wastage, and "Maximum open no." indicates the average maximum number of open items in the solution

Prob. no.	EP without contiguity				EP with contiguity			
	Stocks used	Total wastage	Stocks *w*/ wastage	Maximum open no.	Stocks used	Total wastage	Stocks *w*/ wastage	Maximum open no.
1	10.06	0	0	4.82	9.98	0	0	2.00
2	27.04	0	0	7.86	27.52	0	0	2.84
3	25.06	0	0	7.90	26.4	0	0	3.00
4	23.1	0	0	6.98	23.92	0	0	3.52
5	55.65	0	0	16.54	55.6	1	0.02	6.22
1a	9	3	2	4.72	9	3	2	2.00
2a	23	13	4	7.86	23	13	4.08	2.30
3a	15	0	0	7.90	15	0	0	4.26
4a	19	11	1.02	7.00	19	11	1.64	3.84
5a	53.12	11966	22.8	16.60	53	11450	23.56	6.88

column of "Maximum open no." indicates the maximum number of open items generated by our EP algorithm. It is interesting to note how these numbers were reduced substantially (roughly by half) when we move from CSPs without contiguity to CSPs with contiguity. However, the number of stocks with wastage did not increase very much as a result of considering contiguity. In a sense, three different objectives were considered simultaneously in the EP algorithm although we did not formulate the problem as a multi-objective optimization problem. Appendix B gives some example solutions evolved by the EP algorithm.

To further evaluate the performance of our EP algorithm, we have tested it on ten more problems, i.e., problems 6–10 and 6a to 10a, and compared it against the two-swap algorithm. To make a fair comparison between the EP algorithm and the two-swap heuristic algorithm, the heuristic algorithm was given the same number of cost function evaluations as that used by the EP algorithm in our experiments. Table 4 shows the results of these two algorithms on the ten problems without contiguity.

According to the results in the upper half of Table 4 where fitness values as defined by Eq. (9) were compared, the EP algorithm performed significantly better than the two-swap algorithm for all the problems with multiple stock lengths (problems 6–10), but performed worse on two out of five problems (problems 6a to 10a) with a single stock length. However, according to the total wastage and the total number of stocks used, given in the lower half of Table 4, the EP algorithm outperformed the two-swap algorithm consistently for all ten problems regardless of a single or multiple stock lengths. It performed worse than the two-swap algorithm for three out of the ten problems only according to the number of stocks with wastage. The results in Table 4 also show that the EP algorithm was more robust and reliable than the two-swap algorithm since it had smaller standard deviations over 50 independent runs.

Table 4
The mean best results of the EP and two-swap algorithms *without* contiguity for problems 6 to 10 and 6a to 10a over 50 independent runs

Prob. no.	Req. item	Gen.	EP without contiguity		Two-swap algorithm		EP-TS t-test
			Mean	Std Dev	Mean	Std Dev	
6	200	2000	2.18e − 4	4.99e − 4	1.95e − 2	6.76e − 3	− 20.1146[a]
7	200	2000	2.62e − 3	1.89e − 5	6.78e − 3	4.26e − 3	− 6.9050[a]
8	400	5000	3.10e − 3	3.33e − 3	2.58e − 2	7.97e − 3	− 18.5829[a]
9	400	3000	1.54e − 3	1.58e − 3	6.11e − 3	2.63e − 3	− 10.5325[a]
10	600	10000	2.59e − 2	5.93e − 3	2.83e − 2	5.80e − 3	− 2.0459[a]
6a	200	5000	1.11e − 1	1.05e − 2	1.44e − 1	1.67e − 2	− 11.8289[a]
7a	200	5000	4.71e − 2	9.10e − 3	9.83e − 2	2.30e − 2	− 14.6368[a]
8a	400	8000	1.16e − 1	8.01e − 3	1.07e − 1	1.58e − 2	3.5925[a]
9a	400	10000	9.97e − 2	8.55e − 3	9.66e − 2	1.30e − 2	1.4088
10a	600	20000	1.01e − 1	7.31e − 3	7.71e − 2	1.16e − 2	12.3256[a]

Prob. no.	Req. item	EP without contiguity			Two-swap algorithm		
		Stocks used	Total wastage	Stocks w/ wastage	Stocks used	Total wastage	Stocks w/ wastage
6	200	86.72	0.16	0.16	90.20	21.08	12.02
7	200	73.94	4.00	1.00	76.70	10.40	3.16
8	400	151.10	17.80	1.96	155.74	215.70	12.74
9	400	165.44	3.70	1.96	171.00	26.60	6.72
10	600	228.92	208.20	24.62	234.88	260.90	31.08
6a	200	81.40	309.40	29.96	83.48	488.28	35.02
7a	200	68.88	189.60	7.48	71.18	465.60	14.74
8a	400	148.80	788.00	56.24	150.10	944.00	45.8
9a	400	154.90	730.00	48.54	156.78	955.60	39.40
10a	600	223.56	1037.20	73.06	223.86	1073.20	44.42

[a]The t-value with 49 degrees of freedom is statistically significant at $\alpha = 0.05$.

Similar conclusions can be drawn if we compare the EP and the heuristic algorithms on CSPs with contiguity. Table 5 summarizes the results. According to the fitness measure as defined by Eq. (10), the EP algorithm was outperformed by the heuristic algorithm on only one problem (i.e., problem 8a). It consistently performed better than the heuristic algorithm in terms of the number of stocks used, the total wastage and the maximum number of open items.

It is interesting to note the different results obtained by the EP algorithm when different fitness functions, i.e., those with and without contiguity, were used in Tables 4 and 5. It appears that different fitness functions had little impact on the total number of stocks used, while they had a significant impact on the total wastage. When contiguity was considered, the total wastage also came down substantially for some problems. For example, the total wastage was reduced from 730.00 to 432.40 for problem 9a and from 1037.20 to 643.60 for problem 10a. Such differences were much less significant when the heuristic algorithm was used. However, there was no general trend

Table 5
The mean best results of the EP and two-swap algorithms *with* contiguity for problems 6 to 10 and 6a to 10a over 50 independent runs

Prob. no.	Req. item	Gen.	EP with contiguity		Two-swap algorithm		EP-TS t-test
			Mean	Std Dev	Mean	Std Dev	
6	200	3000	1.39e − 2	6.20e − 3	3.67e − 2	6.85e − 3	− 17.4188[a]
7	200	3000	1.55e − 2	3.02e − 3	2.78e − 2	6.60e − 3	− 12.0173[a]
8	400	5000	3.24e − 2	6.89e − 3	4.66e − 2	7.09e − 3	− 10.1437[a]
9	400	5000	1.04e − 2	2.31e − 3	1.38e − 2	3.16e − 3	− 6.1467[a]
10	600	10000	3.62e − 2	5.31e − 3	3.78e − 2	5.68e − 3	− 1.4043
6a	200	5000	1.05e − 1	1.02e − 2	1.56e − 1	1.87e − 2	− 16.8410[a]
7a	200	5000	7.14e − 2	1.11e − 2	1.10e − 1	1.89e − 2	− 12.3759[a]
8a	400	5000	1.34e − 1	9.13e − 3	1.23e − 1	1.27e − 2	4.8200[a]
9a	400	10000	7.89e − 2	7.76e − 3	9.63e − 2	1.42e − 2	− 7.6321[a]
10a	600	20000	8.44e − 2	7.06e − 3	8.57e − 2	1.18e − 2	− 0.6231

Prob. no.	EP with contiguity				Two-swap algorithm			
	Stocks used	Total wastage	Stocks w/ wastage	Maximum open no.	Stocks used	Total wastage	Stocks w/ wastage	Maximum open no.
6	86.78	4.84	2.46	8.32	90.04	24.30	12.16	10.90
7	74.02	4.60	1.88	11.12	76.82	14.50	4.82	13.42
8	151.78	99.30	14.20	16.18	155.24	179.30	10.96	20.32
9	165.52	7.40	3.90	19.46	171.08	20.00	6.24	19.60
10	229.14	176.50	33.92	24.16	234.46	210.20	27.36	28.54
6a	80.76	254.36	33.70	9.32	83.14	459.04	35.56	13.62
7a	68.96	199.20	15.34	11.88	70.86	427.20	15.88	15.42
8a	148.08	701.60	76.68	16.66	149.44	864.80	43.80	21.34
9a	152.42	432.40	48.40	20.10	155.60	814.00	34.12	26.68
10a	220.28	643.60	70.70	24.14	222.60	922.00	38.78	31.88

[a]The t-value with 49 degrees of freedom is statistically significant at $\alpha = 0.05$.

that could be observed. Whether the total wastage would increase or decrease as a result of contiguity consideration is highly problem dependent.

5. Conclusion

Two critical issues in applying evolutionary algorithms to combinatorial optimization problems are problem representation and search operators used under this representation [27]. Over-emphasizing anyone alone can lead to inaccurate conclusions. This paper takes a more holistic view and tries to design an evolutionary algorithm for CSPs based on the best integration between problem representation and search operators applied to this representation. A new EP algorithm using the order-based representation has been proposed. The new algorithm uses two different

types of mutation, i.e., 3PS and SRI, to deal with CSPs with and without contiguity. Two objectives can be optimized simultaneously by our EP algorithm. The EP algorithm does not use any crossover operators and is simple to implement.

In this paper, the distance between two permutation lists is introduced and used in the mutation design. This enables us to arrive at mutation operators with an appropriate search step size for CSPs. The probabilistic selection of cutting points in 3PS also enables us to bias search toward solutions using less wasted stocks without being trapped in a poor local optimum. Our experimental studies have shown that the results obtained by the EP algorithm performed significantly better than GAs and a heuristic algorithm for most benchmark problems we tested.

There are some improvements which can be made to the EP algorithm in the future. For example, rather than using a generational replacement strategy, we can use a non-generational one where only a certain percentage of the whole population will reproduce, e.g., using the idea of continuous EP [28] or steady-state GA [29]. This can increase the efficiency of an evolutionary algorithm in many cases. Since different search operators introduce different search biases, it would be beneficial to have more than two different mutation operators in the EP algorithm. Self-adaptation can be used to evolve the probabilities of applying these mutations.

Appendix A. The benchmark problems

Problem 1. Stock lengths 10, 13, 15.
Problem 1a. Stock length 14.
20 items

Item length	3	4	5	6	7	8	9	10
No. required	5	2	1	2	4	2	1	3

Problem 2. Stock lengths 10, 13, 15.
Problem 2a. Stock length 15.
50 items

Item length	3	4	5	6	7	8	9	10
No. required	4	8	5	7	8	5	5	8

Problem 3. Stock lengths 10, 13, 15, 20, 22, 25.
Problem 3a. Stock length 25.
60 items

Item length	3	4	5	6	7	8	9	10
No. required	6	12	6	5	15	6	4	6

Problem 4. Stock lengths 13, 20, 25.
Problem 4a. Stock length 25.
60 items

Item length	5	6	7	8	9	10	11	12
No. required	7	12	15	7	4	6	8	1

Problem 5. Stock lengths 4300, 4250, 4150, 3950, 3800, 3700, 3550, 3500.
Problem 5a. Stock length 4300.
126 items

Item length	2350	2250	2200	2100	2050	2000	1950	1900	1850
No. required	2	4	4	15	6	11	6	15	13
Item length	1700	1650	1350	1300	1250	1200	1150	1100	1050
No. required	5	2	9	3	6	10	4	8	3

Problem 6. Stock lengths 86, 83, 79, 76, 72.
Problem 6a. Stock length 86.
200 items

Item length	21	23	24	25	26	27	28	29	31
No. required	10	14	10	7	14	4	13	9	5
Item length	33	34	35	37	38	41	42	44	47
No. required	10	13	10	11	15	12	15	15	13

Problem 7. Stock lengths 120, 115, 110, 105, 100.
Problem 7a. Stock length 120.
200 items

Item length	22	26	27	28	29	30	31	32	34	36	37	38
No. required	6	3	14	12	9	15	11	10	11	13	4	3
Item length	39	46	47	48	52	53	54	56	58	60	63	64
No. required	6	14	7	3	14	9	7	3	5	14	4	3

Problem 8. Stock lengths 120, 115, 110.
Problem 8a. Stock length 120.
400 items

Item length	22	23	24	26	27	28	29	30	31	36	39	41
No. required	12	8	27	15	25	7	10	22	5	16	19	21
Item length	42	48	49	50	51	54	55	56	59	60	66	67
No. required	26	16	12	26	20	25	9	17	22	14	17	9

Problem 9. Stock lengths 120, 115, 110, 105, 100.
Problem 9a. Stock length 120.
400 items

Item length	21	22	24	25	27	29	30	31	32	33	34	35
No. required	13	15	7	5	9	9	3	15	18	17	4	17
Item length	38	39	42	44	45	46	47	48	49	50	51	52
No. required	20	9	4	19	9	12	15	3	20	14	15	6
Item length	53	54	55	56	57	59	60	61	63	65	66	67
No. required	4	7	5	19	19	6	3	7	20	5	10	17

Problem 10. Stock lengths 120, 115, 110.
Problem 10a. Stock length 120.
600 items

Item length	21	22	23	24	25	27	28	29	30	31	33	35
No. required	13	19	24	20	23	24	15	5	24	16	12	24
Item length	36	39	40	41	42	43	44	45	46	47	48	50
No. required	16	4	20	24	6	14	21	20	24	2	11	26
Item length	51	54	56	57	58	61	62	63	64	65	66	67
No. required	23	25	8	16	10	14	6	19	18	11	27	16

Appendix B. The solution examples

Each solution lists the stocks by lines. The numbers delimited by ":" are the requested items. "waste $w_j|L_j$" means that w_j is the wastage of the jth stock and L_j is the stock length of the jth stock. "open o_j" means that o_j is the number of partially finished (open) orders up to the jth stock. The last line of each solution summarizes the overall result and the cost value.

Problem 4 without contiguity

```
7:6:          waste 0¦13, open 2
8:7:5:        waste 0¦20, open 4
11:7:7:       waste 0¦25, open 5
6:6:6:7:      waste 0¦25, open 5
10:8:7:       waste 0¦25, open 6
6:12:7:       waste 0¦25, open 6
7:11:7:       waste 0¦25, open 6
7:6:          waste 0¦13, open 6
8:5:          waste 0¦13, open 6
5:8:          waste 0¦13, open 6
7:6:          waste 0¦13, open 6
9:11:         waste 0¦20, open 7
6:7:          waste 0¦13, open 7
11:7:7:       waste 0¦25, open 7
10:5:10:      waste 0¦25, open 7
6:7:          waste 0¦13, open 6
6:9:10:       waste 0¦25, open 6
5:8:          waste 0¦13, open 6
5:10:10:      waste 0¦25, open 5
11:8:6:       waste 0¦25, open 5
11:5:9:       waste 0¦25, open 4
8:11:6:       waste 0¦25, open 2
11:9:         waste 0¦20, open 0

Total stock wasted ¦ used = 0¦23, total waste = 0, cost = 0
```

Problem 4 with contiguity

```
10:10:      waste 0|20, open 1
10:10:      waste 0|20, open 1
10:10:      waste 0|20, open 0
5:8:        waste 0|13, open 2
5:8:        waste 0|13, open 2
5:8:        waste 0|13, open 2
5:8:        waste 0|13, open 2
8:5:        waste 0|13, open 2
8:5:        waste 0|13, open 2
5:7:8:      waste 0|20, open 1
6:7:        waste 0|13, open 2
12:7:6:     waste 0|25, open 2
6:6:7:6:    waste 0|25, open 2
6:6:7:6:    waste 0|25, open 2
6:6:7:6:    waste 0|25, open 2
6:7:        waste 0|13, open 1
7:7:11:     waste 0|25, open 2
7:7:11:     waste 0|25, open 2
7:11:7:     waste 0|25, open 2
7:11:7:     waste 0|25, open 1
9:11:       waste 0|20, open 2
9:11:       waste 0|20, open 2
11:9:       waste 0|20, open 2
9:11:       waste 0|20, open 0

Total stock wasted|used = 0|24, total waste = 0, cost = 0.01398
```

Problem 4a without contiguity

```
7:11:7:     waste 0|25, open 2
7:7:5:6:    waste 0|25, open 4
6:11:8:     waste 0|25, open 5
11:7:7:     waste 0|25, open 5
7:10:8:     waste 0|25, open 6
6:6:7:6:    waste 0|25, open 6
7:7:        waste 11|25, open 6
12:7:6:     waste 0|25, open 6
10:9:6:     waste 0|25, open 7
10:5:5:5:   waste 0|25, open 7
11:6:8:     waste 0|25, open 7
9:5:11:     waste 0|25, open 7
10:9:6:     waste 0|25, open 7
10:7:8:     waste 0|25, open 7
8:10:7:     waste 0|25, open 6
```

```
7:7:11:      waste 0¦25, open 5
8:11:6:      waste 0¦25, open 5
5:6:9:5:     waste 0¦25, open 3
11:8:6:      waste 0¦25, open 0

Total stock wasted¦used = 1¦19, total waste = 11, cost = 0.03580
```

Problem 4a with contiguity

```
10:5:10:     waste 0¦25, open 2
5:10:10:     waste 0¦25, open 2
10:5:10:     waste 0¦25, open 1
5:5:5:       waste 10¦25, open 1
11:9:5:      waste 0¦25, open 2
8:9:8:       waste 0¦25, open 3
8:9:8:       waste 0¦25, open 3
9:8:8:       waste 0¦25, open 2
11:6:8:      waste 0¦25, open 2
6:6:6:6:     waste 1¦25, open 2
7:6:12:      waste 0¦25, open 3
6:6:7:6:     waste 0¦25, open 3
6:7:6:6:     waste 0¦25, open 2
11:7:7:      waste 0¦25, open 2
7:7:11:      waste 0¦25, open 2
7:11:7:      waste 0¦25, open 2
7:11:7:      waste 0¦25, open 2
11:7:7:      waste 0¦25, open 2
7:7:11:      waste 0¦25, open 0

Total stock wasted¦used = 2¦19, total waste = 11, cost = 0.05309
```

References

[1] Dyckhoff H. A typology of cutting and packing problems. European Journal of Operational Research 1990;44:145–59.

[2] Dyckhoff H, Finke U. Cutting and packing in production and distribution: a typology and bibliography. Heidelberg: Physica-Verlag, 1992.

[3] Sweeney P, Paternoster E. One-dimensional cutting stock decision packing problems. Journal of the Operational Research Society 1992;43(7):691–706.

[4] Roodman G. Near-optimal solutions to one-dimensional cutting stock problems. Computers and Operations Research 1986;13:713–9.

[5] Sinuany–Stern Z, Weiner I. The one dimensional cutting stock problem using two objectives. Journal of the Operational Research Society 1994;45(2):231–6.

[6] Hinterding R, Khan L. Genetic algorithms for cutting stock problems: with and without contiguity. In: Yao X, editor. Progress in evolutionary computation. Lecture notes in artificial intelligence, vol. 956. Berlin: Springer, 1995. p. 166–86.

[7] Yuen B. Heuristics for sequencing cutting patterns. European Journal of Operational Research 1991;55:183–90.

[8] Golden BL. Approaches to cutting stock problem. AIIE Transaction 1976;8:265–74.
[9] Hinxman AI. The trim-loss and assortment problems: a survey. European Journal of Operational Research 1980;5:8–18.
[10] Haessler R, Sweeney P. Cutting stock problems and solution procedures. European Journal of Operational Research 1991;54:141–50.
[11] Bäck T. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.. New York: Oxford University, 1996.
[12] Goldberg D. Genetic Algorithms in search, optimization, and machine learning.. Reading, MA: Addison-Wesley, 1989.
[13] Fogel D. Evolutionary computation: toward a new philosophy of machine intelligence.. New York: IEEE Press, 1995.
[14] Prosser P. A hybrid genetic algorithm for pallet loading. In: Proceedings of the Eighth European Conference on Artificial Intelligence (ECAI-88), Aulander, NC: Pitman, 1988. p. 159–64.
[15] Juliff K. A multi-chromosome genetic algorithm for pallet loading. In: Forrest S, editor. Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA'93), San Mateo, CA, Los Altos: Morgan Kaufmann, 1993. p. 467–73.
[16] Bilchev G. Evolutionary metaphors for the bin packing problem. In: Fogel L, Angeline P, Bäck T, editors. Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming, Cambridge, MA, Cambridge: MIT Press, 1996. p. 333–41.
[17] Reeves C. Hybrid genetic algorithms for bin-packing and related problems. Annals of Operations Research 1996;63:371–96.
[18] Falkenauer E. A hybrid grouping genetic algorithm for bin-packing. Journal of Heuristics 1996;2(1):5–30.
[19] Fogel D. Applying evolutionary programming to selected control problems. Cybernetics and Systems 1993;24:27–36.
[20] Chellapilla K, Fogel D. Exploring self-adaptive methods to improve the efficiency of generating approximate solutions to traveling salesman problems using evolutionary programming. In: Angeline P, Reynolds R, McDonnell J, Eberhart R, editors. Evolutionary programming VI: Proceedings of the Sixth Annual Conference on Evolutionary Programming. Lecture notes in computer science, vol. 1213. Berlin: Springer, 1997. p. 361–71.
[21] Fogel D. Applying evolutionary programming to selected control problems. Computers & Mathematics with Applications 1994;27(11):89–104.
[22] Fogel D. A comparison of evolutionary programming and genetic algorithms on selected constrained optimization problems. Simulation 1995;64(6):397–404.
[23] Yao X. Simulated annealing with extended neighbourhood. International Journal of Computer Mathematics 1991;40:169–89.
[24] Yao X. Comparison of different neighbourhood sizes in simulated annealing. In: Leong P, Jabri M, editors. Proceedings of the Fourth Australian Conference on Neural Networks, Melbourne, Australia, 1993. p. 216–19.
[25] Davis L, editor. Handbook of genetic algorithms, New York: Van Nostrand Reinhold, 1991.
[26] Gilmore P, Gomory R. A linear programming approach to the cutting stock problem—Part II. Operations Research 1963;11:863–88.
[27] Yao X, editor. Evolutionary computation: theory and applications, Singapore: World Scientific Publ. Co. 1999.
[28] Fogel G, Fogel D. Continuous evolutionary programming: analysis and experiments. Cybernetics and Systems 1995;26:79–90.
[29] Syswerda G. A study of reproduction in generational and steady state genetic algorithms. In: Rawlins GJE, editor. Foundations of Genetic Algorithms, San Mateo, CA, Los Altos: Morgan Kaufmann, 1991. p. 94–101.