Fourth International Conference on Natural Computation

An Ant Colony Optimization Algorithm for the One-dimensional Cutting Stock Problem with Multiple Stock Lengths

Qiang Lu,
Department of Computer Science and Technology, Chinese University of Petroleum, Beijing 102249, China
luqiang@cup.edu.cn

Zhiguang Wang
Department of Computer Science and Technology, Chinese University of Petroleum, Beijing 102249, China
wangzhiguang@cup.edu.cn

Ming Chen
Department of Computer Science and Technology, Chinese University of Petroleum, Beijing 102249, China
chenming@cup.edu.cn

Abstract

The Cutting Stock Problem (CSP) with multiple stock lengths is the NP-hard combinatorial optimization problem. In recent years, the CSP is researched by applying evolutionary approaches which includes Genetic Algorithm, Evolutionary Programming, et al. In the paper, an ant colony optimiation (ACO) algorithm for one-dimensional cutting stock problems with muliple stock lengths(MCSP) is presented, and mutation operatation is imported into the ACO in order to avoid the phenomenon of precocity and stagnation emerging. Based on the analysis of the algorithm, the ACO for MCSP has the same time complexity as CSP. Throught exmperiments study, the outcome shows that, compared with other algorithm, the algorithm takes a great improvement in the convergent speed and result optimization.

1. Introduction

The Cutting and Packing problems, which are import class of combinational optimization problems[1], are of wide research in variants practical applications. The one-dimensional traditional Cutting Stock Problem (CSP) is that smaller lengths of items are to be cut from a minimum number of identical stock pieces, and the goal of CSP is to minimize the trim loss. If there is more than one stock length from which items are cut, the problems is called multiple stock length CSP (MCSP).

Traditional approaches to one-dimensional CSPs (CSP and MCSP) can be divided into two categories, linear programming (LP) and heuristics–based method. The LP methods for CSP are first proposed by some researchers [2][3][4]. Gillmore[2][3] develop analytical methods for single size stock and multiple stock lengths. The Claudio approach[4] solves the continuous relaxation of MCSP by means of column generation techniques and applies heuristic methods of branch-and-bound based on LP. LP-based methods have difficulties in dealing with CSPs in which there are large number of variants about the number of stock and its lengths and the number of demand item and its length, because the method can not get the available solutions in acceptable durative time. As a result, heuristic methods have become increasingly popular in solving real-world CSPs. Genetic Algorithm (GA) is introduced to solve CSP by some researchers[5][6][7]. The hybrid grouping GA[5] combines a grouping based GA with a simple local search, and the Exon Shuffling Crossover GA[6] utilities the cost value of individual sub-solutions to execute the crossover. The two above GAs can get some better solutions, but they are only concerned on Bin Packing. The Hinterding's GA[7] extends Bin Packing to MCSP and uses group based encoder/decoder, where each chromosome represents a number of groups of items and each gene represents a group item, to represent the MCSP. Ko-Hsin[8] proposed a evolutionary algorithm (EA) with two new mutation for CSP with or without contiguity based on swap mutations.

Ant colony algorithm, enlightened by ants' food-searching action, is first proposed by Italian scholars M. Dorigo[9]. In the beginning of ACO, some "ants" are initialized, then, each of them searches route according to its heuristic algorithm depositing the pheromone, which is updated by the rate of volatilization on the path of ants creeping. Each ant selects the path according to its heuristic algorithm and the number of pheromone on the trail. With the algorithm running, more and more ants will assemble on the route of larger number of pheromone. The ACO is researched and developed on wide scope of applications, for example Travel Salemen Problem and network route. Levine[10] introduces an ACO with iterated local search, where each ant's solution is improved by moving some of the items around, to deal with Bin-Packing and CSP.

In the article, a novel ant colony optimization algorithm to the MCSP (ACO-MCSP) is proposed. For the ACO-MCSP, each category of ants is as one kind of stock and each category of food is as one kind of demand item length, and each ant do its beast to earch foods within its maximum capacity of holding foods. So the MCSP can be converted into minimizing remained capacity of all ants which have completed searching foods. To reduce the rate of the algorithm running into precocity and stagnation, a mutation strategy is imported.

2. The cutting stock problem

The CSP and MCSP are strongly NP-hard and can are modeled as an integer linear optimization problem. The model of MCSP is described as follow[7].

Object Minimize:

$$W = \sum_{j\in J} w_j x_j \quad (1)$$

Subject to :

$$\sum_{j\in J} a_{ij} x_j = N_i \quad \text{for } i = 1,2,3\ldots n \quad (2)$$

$x_j \geqslant 0$, interger for j∈J

Where, j is pattern of cutting stock, n=number of orders, w_j=waste per run of pattern j (formula (3)), a_{ij}= number of pieces of item i in the pattern j, x_j= number of runs of pattern j, N_i = number of pieces of item i, J = all patterns of cutting stock,

And $$w_j = L_j - \sum_{i=1}^{n} a_{ij} l_i \quad (3)$$

Where, L_i = the length of stock j, l_i = then length of item i.If there is only one stock length L, and l_i is the length of item i, then

$$L = \sum_{i=1}^{n} a_{ij} l_i + w_j \quad , \text{ for all } j \in J \quad (4)$$

Then CSP can be written as :

Min $$X = \sum_{j\in J} x_j \quad (5)$$

From the above formulas, the CSP is the special instance of MCSP. However, the CSP time complexity is as the same as the MCSP's, because they both include the variable L_j, l_j and a_{ij}, which are the scale of both CSPs related to the variable i and j, and the number of variable i and j are the number of stocks and demands items respectively.

3. The Model of ACO-MCSP

3.1. Transformation from MCSP to ACO

In ACO, the ant_α's maximal load of holding foods is the variable L_α, the weight of $food_j$ is the variable l_j in the MCSP, the number of ant of searching foods is variable J and the number of $food_j$ is x_j, the number of $food_j$ in ant$_i$ is α_{ij}. Each ant_i does its beast to search foods Q within its max capacity of holding food ($L_i - \sum_{i\in Q} l_i > 0$). When every ant has completed searching foods and there is no existing foods leaved behind, the ants of one generation have completed the task of searching foods. So, the problem of the ant colony pursue the maximized foods weight is equal to the MCSP of minimizing trim loss.

The ant_α's path is shaped by the ordered food that it has find. If the ant holds more weight of foods based on its capacity, the speed of ant_α crawling will become slower and the $pheromone_\alpha$ that ant_α deposites on the path become denser. The denseness of $pheromone_\alpha$ between $food_i$ and $food_j$ represent the prior probability which ant_α selecting the $food_j$ after it has found $food_i$ or selecting the $food_i$ after found $food_j$. Everyone in ant_α searches food based on its surround foods and the $pheromone_\alpha$ which its kin ants leave behind. Therefore, more and more ant_αs accumulate on the path where more pheromone desposites, and the ant_αs has trend to select foods on the path.

In order to search food collaboratively with each other, eah ant in one generation must share a *tabu* of foods. After a ant find one $food_j$, the $food_j$ will be deteled from the *tabu*. When there is no food in *tabu*, the task of ant searching food will finish. At the time of ants completed searching food, the pheromone that ant desposited on the path will be update by the result of its food.

To avoid precocity and stagnation in the process of searching food, a mutation strategy is added to result of ant colony found foods. Ant the pheromone will be updated based on the result of mutation. The detail algorithm of ACO-MCSP is represented as follow.

3.2. The ACO-MCSP representation

Step 1. Initialization.

At the beginning of the algorithm running, some parameters are initializaiton. Set g:=1 (g is the counter of ant's generation), initiate food tabu and pheromone vector τ , select randomly searching-food taskant from ant colony based on the following AllocTaskAnt algorithm.

In the algorithm, vector F is the container of ordered foods which are sorted from high to low based on the weight of food. F_i is the weight of $food_i$, and $|F_i|$ is the number of $food_i$. Vector Ant is the all ants in the ant colony, and Ant_{ak} is the maximal capacity of the kth ant_a holding food. Vector *taskant* is the container of searching-food ants.

```
allocTaskAnt（F,Ant）： taskAnt
        s1. length=sum(F)
        s2. if |Ant| ≤ 0 then return taskAnt
                  ant_ij=randomSelect(Ant)
            l=ant_ij
            task=fasle;
        s3. f = takeFormHead (F)
            if l>f then
              l=l-f
              task=true
              goto s3
            else
        if taks =true then
          length=length- $ant_{ij}$
          TaskAnt.add ($ant_{ij}$)
        else
          goto s2
```

Step 2. Searching and Selecting Foods

After the ant_{ak} found $food_i$, at the time t, the probrablityof $food_j$ selected by the ant ($P_{ij}^{\alpha k}(t)$) is related to the following formular (6) (7).

$$P_{ij}^{\alpha k}(t)=\begin{cases}\dfrac{[\tau_{ij}^{\alpha}(t)]^u\times[\eta_{ij}^{\alpha k}(t)]^v}{\sum\limits_{s\in tabu}[\tau_{is}^{\alpha}(t)]^u\times[\eta_{is}^{\alpha k}(t)]^v} & tabu_j\neq 0\\ 0 & tabu_j=0\end{cases}\qquad(6)$$

Where, $tabu_j$ is the number of $food_j$, u and v are parameters which can denote respectively pheromone accumulation and the effect from heuristic function($\eta_{ij}^{\alpha k}(t)$) to transition probability. The heuristic function $\eta_{ij}^{\alpha k}(t)$ represents the expectation of ant_a selecting $food_j$ at the time t.

$$\eta_{ij}^{\alpha k}(t)=\begin{cases}\dfrac{1}{ant_{\alpha k}-V_{\alpha k}(t)-F_j} & ant_{\alpha k}-V_{\alpha k}-F_j>0\\ \infty & ant_{\alpha k}-V_{\alpha k}-F_j=0\\ 0 & ant_{\alpha k}-V_{\alpha k}-F_j<0\end{cases}\qquad(7)$$

Where, the vector $V_{\alpha k}$ is the sum of food weight in the $ant_{\alpha k}$, the formular $ant_{\alpha k}-V_{\alpha k}-F_j$ represents the remaider load of food. The litter value of the formular, earilier task of the ant_{ak} searching food will be completed. If the value is 0, the ant_{ak} get the best food and finish its task. If the value is less than 0, the $food_j$ is not fit to the ant_{ak}..

At the time t, the ant_{ak} selects one $food_j$ throught the method of roulette based on $P_{ij}^{\alpha k}(t)$. After the ant_{ak} selected the $food_j$, the $food_j$ is inserted into $V_{\alpha k}$ and is deleted from $tabu_j$.

Step 3. Compute the result of ant searching food

After the ants in taskant have done its best to get the foods based on their capacity, the efficiency of each ant searching food is computed by the formula (8)

$$e_{\alpha k}=\frac{V_{\alpha k}}{ant_{\alpha k}}\qquad(8)$$

$$d(g)=\sum_{\alpha k\in taskAnt}e_{\alpha k}\qquad(9)$$

The formula $d(g)$ is the sum efficiency of the ants in g generation, if $d(g)=|taskant|$, the ants will get the best solution and the algorithm finish.

To avoid the phenomenon of precocity and stagnation emerging in ACO algorithm, a mutation strategy is imported to the algorithm. To assert the mutatation operation wether run or not, we define convergence to denote the status of ACO algorithm.

Def 1. Define convergence in ant colony as equation (10) and (11) does,

$$Con(g)=D(g)-\frac{\sum_{k=1}^{g-1}D(k)}{g-1}\qquad(10)$$

$$D(g)=\frac{d(g)}{n}\qquad(11)$$

Where g denotes the generation of ACO algorithm running, $D(g)$ denotes the average efficiency of ants searching result. From equation (10), $Con(g)$ will become small when the speed of convergence tend to slow or stop. So the convergence in ant colony represents the trend of ACO searching result.

If $|Con(g)|$ < C, execute the following mutation operation, otherwise goto step 5.

Step 4. Mutation

For each ant in *taskAnt*, delete randomly one food in its V and insert the food deleted into vector M. Then, select randomly one food from M and insert it into the ant which havs enough capacity to the food. Until there is not fit food to be inserted into ant from *taskAnt*, the mutation opertaion finish.

Step 5. Update pheromone trail

Each ant_a in taskAnt will deposite the same $pheromone_a$ on the trail based on the order of foods that the ant found。The $pheromone_a$ is updated by the formula (12).

$$\tau_{ij}^{\alpha}(g+1)=(1-\rho)\times\tau_{ij}^{\alpha}(g)+\triangle\tau_{ij}^{\alpha}(g)\qquad(12)$$

Where, 1-ρ denotes trail evaporation (0<ρ<1), $\triangle\tau_{ij}^{\alpha}(g)$ represent the ant_a deposites the pheromone between $food_i$ and $food_j$ at the genearation g.

$$\triangle\tau_{ij}^{\alpha}(g)=\begin{cases}\dfrac{Q}{\sum\limits_{k\in taskAnt_{\alpha}}(ant_{\alpha k}-V_{\alpha k})} & F_i \text{ and } F_j \text{ are sequence and belong to } V_{\alpha k}\\ 0 & otherwise\end{cases}\qquad(13)$$

Where, the const value Q is the pheromone intensity that all ants in taskant leave behind.

Step 6. Generate next generation taskant and tabu.

Based on the foods of ants seaching, the *tabu* is initialized. And the taskAnt can be got form the *AllocTaskAnt* algorithm. Then, set g:=g+1 and goto step 2

3.3. Analysis of the Algorathm

Suppose n is the number of food, and m is the number of stock, $t(f)$ is the number of food category, $t(s)$ is the number of stock category, and $t(s,f)$ is the number of taskant which are got by *AllocTaskAnt* algorithm. The time complexity (TC) of *tabu* initialzing is $O(t(s,f).t(s))$, the TC of *AllocTaskAnt* algorithm is $O(m.n)$, the TC of vector *pheromone* is $O((t(f))^2.t(s))$. So, the TC of initailized stage is $O(t(s,f)+m.n+(t(f))^2.t(s))$.

On the step 2, the TC of ant selecting one food based on heuristic function and pheromone is $O(t(s,f).(t(f))^2)$, and the TC of roulette method is $O(t(f))$. On the step 3, the TC of computing the efficientcy of ant searching foods is $O(t(s,f))$,and the TC of convergence computing is $O(1)$. On the step 4, the TC of mutation is $O((t(s,f))^2)$. On the step 5, the TC of pheromone updating is $O((t(s,f)).\ (t(f))^2)$.

The TC of step1 can be ignored, because the steps include initialized operations and the step does not join in the iterative cycle of the algorithm. In each cycle of the algorithm, the TC of ACO-MCSP is

$$O(t(s,f).(t(f))^2+\ t(f)+\ t(s,f)+\ (t(s,f))^2+(t(s,f)).(t(f))^2)+m.n) \approx O(t(s,f).(t(f))^2+m.n+\ (t(s,f))^2). \quad (14)$$

Through the above analysis, the m,n and $t(f)$ between CSP and MCSP are same. Based on the *AllocTaskAnt* algorithm, the $t(s,f)$ is related to varible n, stock lengths and demand item lengths.So the TC of ACO for CSP is as the same as for MCSP.

4. Experimental result

For the sake of validating the capability of ACO-MCSP algorithms, we, with C++ programme, implemented respectively the pure ACO (not mutation), the Hinterding's GA[7] and the algorithm (ACO-MCSP) program. To evaluate the result of these algorithms, define the trim loss rate as following fomula(15)[7].

$$waste = \frac{1}{n}\left(\sum_{i\in 1,n}\sqrt{\frac{waste_i}{stocklength_i}} + \frac{number\_wasted}{n}\right) \quad (15)$$

Where, n = number of groups, $stocklength_i$ = the stock i's length will be cut from, wastei = $stocklength_i$ - sum of items in stock i, *number_wasted* = number of stock lengths with wastage.

We use the problems (1-5 and 7,8, 10)[8] to compare our approach to the others. And we did 20 independent test runs for each problem with CSP without contiguity[7]. We set the ρ=0.75, initialized pheromone is 0.20, and the const Q = *min*(*item*)/num(*minitems*), for example the Q= 3/5=0.6 in problem 1.

The datas in table1 are the result of GA[7], Pure ACO and ACO-MCSP for MCSP without contiguity. The "Req item" in table 1 is the total number of requested items, "Gen" is the maximal number of generations, "waste" is value are computed by formula(15), "found" indicates the generation number when the best solution was found. From the datas in table1, we can conclude than the ACO-MCSP can get the better (equal) result than the GA, and the number of ACO-MCSP's generations is smaller than GA or Pure-ACO. Although Pure-ACO can get its best result, the result is worst in the three algorithms because precocity and stagnation emerges.

The datas in table 2 are the result of the above thress algorithms for CSP without continguity. From the results, we can conclude that the Pure-ACO is not fit to the CSP because phenomena of precocity and stagnation emerges is high frequency. Ant the ACO-MCSP can get its best result which is close to the GA's, but the time of ACO-MCSP researching result is higher than the time of GA。

In table 3, "avg" indicates how many stocks were used on average, "best" indicates the number of stocks in the best solution found, and "time" indicates the average running time of best result in CPU seconds. From data the table3 and the figure1, we can conclude that GA and ACO-MCSP can get same or approximate result. But the MCSP can get the best solution with higher convergence speed.

Table 1. Comparison waste and found between GA, Pure-ACO and ACO-MCSP for MCSP

Prob	Req item	GA			Pure-ACO			ACO-MCSP		
		Gen	waste	Found	Gen	waste	Found	Gen	waste	Found
1	20	1000	1.0	407	1000	1.0	507	1000	1.0	482
2	50	2000	1.0	740	2000	0.9957	661	2000	1.0	506
3	60	2000	1.0	407	2000	0.9972	683	2000	1.0	512
4	60	3000	0.9995	2294	2000	0.9981	792	2000	0.995	913
5	126	3000	0.9998	2294	2000	0.9871	1600	2000	0.9998	1314
7	200	5000	0.9974	3211	5000	0.9892	2712	5000	0.9982	1500
8	400	5000	0.9979	3713	5000	0.9872	2814	5000	0.9979	2231
10	600	5000	0.9741	4572	5000	0.9321	3002	5000	0.9817	2714

Table 2. Comparison waste and found between GA, Pure-ACO and ACO-MCSP for CSP

Prob	Req item	GA			Pure-ACO			ACO-MCSP		
		Gen	waste	Found	Gen	waste	Found	Gen	waste	Found
1a	20	1000	0.9133	296	1000	0.8056	102	1000	0.9133	201
2a	50	2000	0.9227	1184	2000	0.8912	201	2000	0.9231	578
3a	60	2000	1.0	1184	2000	0.9921	845	2000	1.0	982
4a	60	3000	0.9642	851	2000	0.9113	912	2000	0.9638	993
5a	126	3000	0.8479	2294	2000	0.8312	1201	2000	0.8481	1864

Table 3. Comparion avg, best and time between GA, Pure-ACO and ACO-MCSP for MCSP

Prob	Req item	GA			Pure-ACO			ACO-MCSP		
		avg	best	time	avg	best	Time	avg	best	time
1	20	10.06	10.06	2.30	11.2	10.06	0.91	10.06	10.06	1.23
2	50	27.35	27.18	42.12	29.14	28.17	3.12	27.18	27.04	27.84
3	60	26.01	26.01	59.56	27.81	27.81	19.71	25.73	25.09	59.13
4	60	23.10	23.10	44.17	24.51	24.13	31.32	23.10	23.10	56.69
5	126	56.01	55.87	107.81	57.94	56.48	98.67	55.97	55.65	87.12
7	200	74.57	74.02	213.58	82.19	79.32	241.12	75.82	74.02	183.12
8	400	151.92	151.74	661.43	179.07	163.78	567.25	153.57	151.66	451.78
10	600	230.08	229.57	2821.78	245.79	234.13	2017.78	230.45	229.14	1992.5

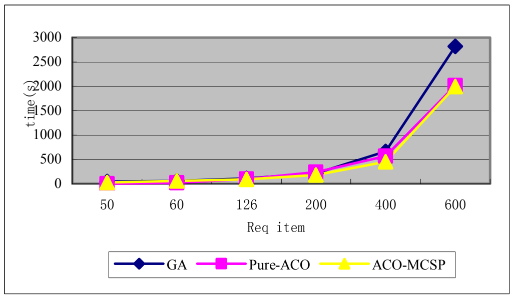

Figure 1. Comparsion the running time with differen req items between GA, Pure-ACO and ACO-MCSP for MCSP

5. Conclusion

The paper has presented an ACO approach for the cutting stock problem with multiple stock lengths. First, the MCSP is converted into the ACO based on the formulas of integer linear optimization. Second, the pheromone trail is built to represent the prior probability of selecting food, and the artificial ants search foods using a combination of heuristic information and pheromone trail. Third, when all ants have completed its task in one generation, the efficiency of each taskant will be computed, and the formular of precocity and stagnation judging is created. If the phenomenon of precocity and stagnation emerge, then the mutation operation will be executed. At last, the pheromone trails are updated by the order foods of ants searching. Throught analyses of the algorithm and the experiment study, the MCSP and CSP have the same time complexity for ACO. The best resolution of ACO-MCSP running is better than (or closed to) the GA. Compared with GA, ACO-MCSP has the higher speed of searching foods for the large scale of require items.

Acknowledgements

We would like to thank Ko-Hsin Liang, Robert Hinterding and John Levine for sharing their results. The current research is partly supported by the Natural Science Foundation of China under Grant No.60072006.

References

[1]. H.A. Dyckhoff, “typology of cutting and packing problems”, *European Journal of Operational Research*, 44(2) 1990, pp.145-159.
[2]. P.C. Gillmore., P.E. Gomony, “A linear programming approcah to the cutting-pack problem”, *Operations Research* 9, 1961,pp.849-859.
[3]. P.C. Gillmore, P.E. Gomony, “A linear programming approcah to the cutting-pack problem (Part II)”, Operations *Research* 11, 1963, pp.863-887.
[4]. A. Claudio, J.M. Valerio de Carvalho, “A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problems”, *Computer&Operations Research* 35, 2008, pp.1315-1328.
[5]. E.Falkenauer, “A hybrid grouping genetic algorithm for bin packing”, *Journal of Heuristics* 2, 1996, pp.5–30.
[6]. P.Rohlfshagen, J.A.Bullinaria, “A genetic algorithm with exon shuffling crossover for hard bin packing problems”, *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, London, England, 2007, pp.1365-1371.
[7]. R. Hinterding, L. Khan, “Genetic algorithms for cutting stock problems: with and without contiguity”, *Progress in Evolutionary Computation*, Germany, Berlin, Springer, 1995, pp.166–186,.
[8]. L.Ko-Hsin, Y.Xin, C. Newton, D. Hoffman, “A new evolutionary approach to cutting stock problems with and without contiguity”, *Computer & Operations Research* 29, 2002, pp.1641-1659.
[9]. M. Dorigo, “Optimization, Learning and Natural Algorithms”, *Ph.D.Thesis*, Department of Electronics, Politecnico diMilano, Italy, 1992.
[10]. J.Levine, F. Ducatelle, “Ant Colony Optimisation and Local Search for Bin Packing and Cutting Stock Problems”, *Journal of the Operational Research Society Special Issue on Local Search* 55(7), 2004,, pp.705-716.