Approximate Muscle Guided Beam Search for Three-Index Assignment Problem

He Jiang, Shuwei Zhang, Zhilei Ren, Xiaochen Lai, and Yong Piao

Software School, Dalian University of Technology, Dalian, 116621, China
jianghe@dlut.edu.cn

Abstract. As a well-known NP-hard problem, the Three-Index Assignment Problem (AP3) has attracted lots of research efforts for developing heuristics. However, existing heuristics either obtain less competitive solutions or consume too much time. In this paper, a new heuristic named Approximate Muscle guided Beam Search (AMBS) is developed to achieve a good trade-off between solution quality and running time. By combining the approximate muscle with beam search, the solution space size can be significantly decreased, thus the time for searching the solution can be sharply reduced. Extensive experimental results on the benchmark indicate that the new algorithm is able to obtain solutions with competitive quality and it can be employed on instances with large-scale. Work of this paper not only proposes a new efficient heuristic, but also provides a promising method to improve the efficiency of beam search.

Keywords: Combinatorial Optimization, Heuristic, Muscle, Beam Search.

1 Introduction

The Three-Index Assignment Problem (AP3) was first introduced by Pierskalla [1, 2]. It is a NP-hard problem with wide applications, including addressing a rolling mill, scheduling capital investments, military troop assignment, satellite coverage optimization [1, 2], scheduling teaching practice[3], and production of printed circuit boards [4]. It can be viewed as an optimization problem on a 0-1 programming model:

$$\min \sum_{i\in I}\sum_{j\in J}\sum_{k\in K} c_{ijk}\, x_{ijk} \quad . \tag{1}$$

subject to

$$\sum_{j\in J}\sum_{k\in K} x_{ijk} = 1, \quad \forall i \in I \quad . \tag{2}$$

$$\sum_{i\in I}\sum_{k\in K} x_{ijk} = 1, \quad \forall j \in J \quad . \tag{3}$$

$$\sum_{i\in I}\sum_{j\in J} x_{ijk} = 1, \quad \forall k \in K \quad . \tag{4}$$

$$x_{ijk} \in \{0,1\}, \quad \forall i \in I, j \in J, k \in K \quad . \tag{5}$$

where $I = J = K = \{1,2,3,...,n\}$.

The solution of AP3 can be presented by two permutations:

Y. Tan et al. (Eds.): ICSI 2014, Part I, LNCS 8794, pp. 44–52, 2014.

$$\min \sum_{i}^{n} c_{i,p(i),q(i)} \ , \quad p(i),q(i) \in \pi_N \ . \tag{6}$$

where π_N presents the set of all permutations on the integer set $N=\{1,2,\ldots,N\}$. Here c_{ijk} represents the cost of a triple $(i, j, k) \in I \times J \times K$.

Due to its intractability, lots of exact and heuristic algorithms are proposed to solve it, including Balas and Saltzman [5], Crama and Spieksma[6], Burkard and Rudolf [7], Pardalos and Pitsoulis[8], Voss [9], Aiex, Resende, Pardalos, and Toraldo[10], Huang and Lim [11], Jiang, Xuan, and Zhang [12]. Among these algorithms, LSGA proposed by Huang and Lim [11], and AMGO proposed by Jiang, Xuan, and Zhang [12] perform better than the other heuristics. LSGA can obtain a solution within quite a short time, but on difficult instances LSGA might not perform well in terms of the solution quality, while AMGO can obtain better solutions with high quality, but on large instances the running time is intolerable. It would be ideal to achieve a good trade-off between solution quality and running time.

To tackle the challenges in balancing solution quality and running time, we propose a new heuristic named Approximate Muscle guided Beam Search (AMBS). It combines two phases. In the first phase, a multi-restart local search algorithm is used to generate a smaller search space, which we call "approximate muscle". In the latter phase, beam search is employed to obtain a high quality solution. By combining the approximate muscle and beam search, we can obtain solutions with relatively high quality in a short time. Experimental results on the standard AP3 benchmark indicate that in terms of solution quality, the solutions obtained by AMBS are better than LSGA and not worse than the pure beam search, while in terms of running time, AMBS can deal with large instances that AMGO and the pure beam search cannot.

The rest of this paper is organized as follow. In Section 2, a review of the muscle and beam search is given. In section 3, the framework of AMBS is proposed. Experiment results are reported in Section 4. In Section 5, the conclusion is presented.

2 Muscle and Beam Search

In this section, we present the two concepts the muscle and beam search. For each concept, we first briefly review its related work and then present its details.

2.1 Muscle

The proposition of the concept muscle is inspired by the backbone. The backbone means the shared common parts of optimal solutions for an instance. It is an important tool for NP-hard problem. In contrast to the backbone, the muscle is the union of optimal solutions. It was first proposed by Jiang, Xuan, and Zhang in 2008 [12]. Some efficient algorithms have been proposed using the muscle. For example, Jiang and Chen developed an algorithm for solving the Generalized Minimum Spanning Tree problem with the muscle [13]. Obviously, if the muscle could be obtained, the search space for an instance would be decreased sharply. However, Jiang has proved that there is no polynomial time algorithm to obtain the muscle for AP3 problem [12].

Now that the muscle cannot be obtained directly, there are some other ways to approximate it. Experiments conducted by Jiang indicate that the probability that the union of local optima contains the optimal solution increases with the growth of the number of local optimum, while the size increases slower [12]. Hence, the union of local optima can approximate the muscle, it can be named the approximate muscle.

2.2 Beam Search

Beam search is a widely-used heuristic algorithm. For example, Cazenave combined Nested Monte-Carlo Search with beam search to enhance Nested Monte-Carlo Search [14], López-Ibáñez and Blum combined beam search with ant colony optimization to solve the travelling salesman problem with time windows [15].

Beam search can be viewed as an adaptation of branch-and-bound search. The standard version of beam search builds its search tree using breadth-first search. At each level of the search tree, a heuristic algorithm is employed to estimate all the successors, and then a predetermined number of best nodes are stored, while the others are pruned off permanently. This number is called the beam width. By varying the beam width, beam search varies from greedy search (the beam width equals to 1) to a complete breadth-first search (no limit to the beam width). By limiting the beam width, the complexity of the search becomes polynomial. In this way, beam search can find a solution with relatively high quality within practical time. We call the standard version of beam search as the pure beam search, to distinguish it with AMBS.

3 Approximate Muscle Guided Beam Search for AP3

In this section, we introduce the detail of AMBS. We will first present the framework of our algorithm, then we will show the details in the following subsections.

3.1 AMBS for AP3

The algorithm is shown in Algorithm 1. The instance $AP3(I,J,K,c)$, the number of sampling k, and the beam width *width* are the inputs. The output is the solution s^*. A instance is stored in a 3-dimesional array, and the solutions is stored in two arrays.

Two phases are in the algorithm. In the beginning of the search phase, the order of search level is sorted ascending by the number of triples. When calculating the lower bound of each branch, more time will be consumed when the branch is at the higher level. Thus, after the sorting, fewer nodes are in the higher level, and searching time is reduced. More details about building the search tree is introduced in section 3.3.

In the following subsections, we will discuss the details of two phases, respectively.

3.2 Approximate Muscle for AP3

In the first phase, we use the union of local optima to approximate the muscle. The detail is shown in Algorithm 2. The inputs are an AP3 instance and the number of sampling.

The output includes the approximate muscle a_muscle, and the best solution s is obtained. The approximate muscle is stored in a 3-dimensional array, where the cost is the same value as the instance if it is sampled, or infinite if not.

The approximate muscle is initialized as an empty set first. Then k local optima are obtained to make up the approximate muscle. A random solution is generated, then a local search algorithm is applied to obtain a local optimum. The local search algorithm we use here is the Hungarian local search proposed by Huang and Lim [11]. The best local optimum is recorded.

3.3 Beam Search for AP3

In the second phase, we use beam search to find a better solution. Before the introduction of beam search for AP3, we will first present how we build the breadth-first search tree. An instance of AP3 can be represented as a three-dimensional matrix. First, the matrix is divided into n layers. For example, an instance with the size of 4 is divided into 4 layers. Select a layer to be level 1 of the search tree. Then another layer is selected to build next level. Since one triple has been determined in level 1, there are 9 successors, and 144 nodes in all in level 2. In this way, the tree is built. If using the muscle to build the tree, only the triples in the muscle are considered.

Algorithm 1. AMBS for AP3
Input: AP3instance $AP3(I,J,K,c)$, k, $width$
Output: solution s^*
Begin
//the sampling phase
(1) obtain the approximate muscle a_muscle and a solution s' as the upper bound with $GenerateAM\ (AP3(I,J,K,c),k)$;
//the search phase
(2) sort the search order of the approximate muscle and get the $order$;
(3) obtain the solution s^* with $BS(a\_muscle, width, s', order)$;
End

Algorithm 2. GenerateAM (Generate Approximate Muscle)
Input: AP3instance $AP3(I,J,K,c)$, k
Output: a_muscle, solution s'
Begin
(1) $a\_muscle = \varnothing$
(2) for $counter = 1$ to k do
(3) for $i = 1$ to n do $p[i] = i$, $q[i] = i$;
(4) for $i = 1$ to n do
(5) let j_1, j_2 be two random integers between 1 and n;
(6) swap $p[i]$ and $p[j_1]$; swap $q[i]$ and $q[j_2]$;
(7) let $s = \{(i, p[i], q[i]) \mid 1 \le i \le n\}$;
(8) obtain a local optimum s_{local} by applying the local search to s;
(9) $a\_muscle = a\_mucle \cup s_{local}$;
(10) if $c(s_{local}) < c(s')$ then $s' = s_{local}$
End

The detail of beam search for AP3 is presented in Algorithm 3. The inputs are the approximate muscle *a_muscle*, the beam width *width*, the best local solution *s'*, and the search order *order*. The output is the solution s^* of this AP3 instance.

In the algorithm, a candidate represents a branch to be searched. When the search comes to a certain level, the lower bounds of all the successors are generated. The arrays fp and fq are used to record the determined triples to guarantee the constraints. Then the lower bounds are calculated. The lower bound includes three parts: the sum of the triples' cost in the candidate, the cost of the triple relevant to the successor, and lower bound of the sub-problem. The sub-problem is the approximate muscle without the layers containing the determined triples. The lower bound calculating method is proposed by Kim et al. [16]. In the end, at most *width* successors with smaller lower bound than the cost of *s'* are kept to be the new candidates. After searching, a local search algorithm is employed to the remaining candidates. Then the best (including *s'*) is chosen to be the solution s^*. Since the approximate muscle is stored in the same way as an instance, beam search algorithm can be used to solve AP3 problem independently.

Algorithm 3. BS (Beam Search)
Input: a_muscle, $width$, solution s', $order$
Output: solution s^*
Begin
(1) for every $level$ based on $order$ in the search tree do
(2) for every *candidate* do
(3) for every triple $(i, j, k) \in candidate$ do $fp[j] = true, fq[k] = true$;
(4) for every triple $(order[level], j, k) \in a\_muscle$ do
(5) if $fp[j] = false$ and $fq[k] = false$ then
(6) generate the sub-problem; calculate the lower bounds;
(7) sort the bounds of all *candidate*;
(8) for $i = 1$ to $width$ do
(9) if lower bound of the branch $< c(s')$ then
(10) this branch belongs to the new *candidates*;
(11) else break;
(12) employ the local search algorithm on every *candidate* and choose the best to be the solution s^*
End

4 Experimental Result

In this section, we first show the parameter tuning result. Then we present the results of our algorithm on the benchmark. The codes are implemented with C++ under windows 7 using visual studio 2010 on a computer with Intel Core i3-M330 2.13G. The time in the tables is measured in seconds.

4.1 Parameter Tuning

Two parameters are used in AMBS, the number of sampling and the beam width. We determine the number of sampling as 1000, the same value in AMGO [12]. As for the

beam width, we test different beam widths {100, 200, 300, 400} on 4*5 instances from Balas and Saltzman Dataset (see Section 4.2) and 6 instances from Crama and Spieksma Dataset (see Section 4.3).

Table 1 shows the result of our parameter tuning. We run the algorithm 10 times on each instances of the Balas and Saltzman Dataset with each beam width, while run it once on Crama and Spieksma Dataset since the result varies little. The value of Balas and Saltzman Dataset is the average value of each size. The result indicates that the solution quality and the running time rise with the increase of the beam width. Note that the running time of 3DA99N1, 3DA198N1 and 3D1299N1 vary little in different beam widths. This is because the approximate muscle space of each of the instance is so small. The running time of 3DI198N1 is the longest. When the beam width is 300, the running time is about 20 minutes. In order to balance the quality of the solution and the running time, we determine the beam width as 300 in the rest of experiments.

Table 1. Beam Width Tuning

Instance Id	Width=100		Width=200		Width=300		Width=400	
	Cost	Time	Cost	Time	Cost	Time	Cost	Time
BS_14_x	10	0.74	10	1.07	10	1.31	10	1.72
BS_18_x	6.86	2.48	6.66	4.25	6.48	5.97	6.46	7.82
BS_22_x	4.86	6.75	4.62	12.00	4.34	17.13	4.34	22.45
BS_26_x	2.74	15.01	2.34	27.05	2.1	39.16	2.08	51.01
3DA99N1	1608	7.01	1608	7.39	1608	7.24	1608	7.16
3DA198N1	2662	62.05	2662	65.00	2662	63.15	2662	64.34
3DIJ99N1	4797	16.59	4797	18.53	4797	20.33	4797	22.11
3DI198N1	9685	479.66	9684	863.94	9684	1219.82	9684	1566.40
3D1299N1	133	1.70	133	1.75	133	1.73	133	1.73
3D1198N1	286	169.37	286	276.14	286	383.95	286	573.86

4.2 Balas and Saltzman Dataset

This dataset is generated by Balas and Saltzman[5]. It contains 60 instances with size of 4, 6, 8, ..., 26. For each size, five instances are generated randomly.

Table 2. Balas and Saltzman Dataset (12*5 instances)

n	Opt.	LSGA		AMGO		Beam Search		AMBS	
		Cost	Time	Cost	Time	Cost	Time	Cost	Time
4	42.2	42.2	0	42.2	0.01	42.2	0.01	42.2	0.01
6	40.2	40.2	0.01	40.2	0.03	40.2	0.03	40.2	0.03
8	23.8	23.8	0.03	23.8	0.06	23.8	0.06	23.8	0.06
10	19	19	0.37	19	0.11	19	0.14	19	0.11
12	15.6	15.6	0.87	15.6	0.18	15.6	0.62	15.6	0.26
14	10	10	1.73	10	0.26	10	7.62	10	1.31
16	10	10	1.89	10.16	0.52	10	22.74	10	3.32
18	6.4	7.2	2.95	6.4	0.97	6.4	49.65	6.48	5.97
20	4.8	5.2	4.01	4.8	1.67	4.8	98.18	4.88	10.43
22	4	5.6	4.54	4	6.26	4.24	185.70	4.34	17.13
24	1.8	3.2	5.66	1.96	12.16	2.38	313.60	2.28	26.95
26	1.3	3.6	10.78	1	6.62	2.38	526.59	2.1	39.16

Table 2 shows the result on this dataset. The column "Opt." is the optimal solution reported by Balas and Saltzman[5]. "LSGA" is the result reported in Huang's paper [11] using a PIII 800MHz PC. "AMGO" is the results of the program implemented according to Jiang's paper [12]. "Beam Search" is the result of the pure beam search. "AMBS" is the result of our algorithm. The results of AMGO, the pure beam search and AMBS are the average cost after running 10 times on each instance. The solutions of size 26 found by AMGO are 0,0,2,1,2 respectively, the average cost is 1. Because the average of any five integers cannot be 1.3, it may be a typo in Balas's paper.

The result indicates that AMBS can get solutions with higher quality than LSGA. AMGO generates the best solutions, and the running time is quite short, because it employs a global search on the approximate muscle and the search space is quite small. AMBS uses an incomplete search and needs to estimate the lower bound of each branch, thus, the quality of solutions is a little worse and the running time is longer than AMGO. Compared with the pure beam search, the running time of AMBS is about one-tenth of the pure beam search, but the quality is comparable. This is because that there are much fewer successors in the approximate muscle.

4.3 Crama and Spieksma Dataset

This dataset is generated by Crama and Spieksma[6]. Three types are in the dataset. In each type, three instances have the size of 33, and three have the size of 66.

Table 2 shows the result on this dataset. AMGO, the pure beam search and AMBS are executed once since the result varies little. There are some cells with no value, because the running time is longer than 30 minutes, and we regard it unacceptable.

Table 3. Crama and Spieksma Dataset. (18 instances)

n	Instance Id	LSGA		AMGO		Beam Search		AMBS	
		Cost	Time	Cost	Time	Cost	Time	Cost	Time
33	3DA99N1	1608	0.03	1608	7.60	1608	649.74	1608	7.24
33	3DA99N2	1401	0.11	1401	7.11	1401	1733.90	1401	6.52
33	3DA99N3	1604	0.11	1586	7.61	1604	1606.99	1604	7.30
66	3DA198N1	2662	0.55	2662	71.22	-	-	2662	63.15
66	3DA198N2	2449	0.27	-	-	-	-	2449	74.20
66	3DA198N3	2758	0.58	-	-	-	-	2758	82.07
33	3DIJ99N1	4797	0.11	-	-	-	-	4797	20.33
33	3DIJ99N2	5067	0.26	-	-	-	-	5067	35.95
33	3DIJ99N3	4287	0.26	-	-	-	-	4287	26.07
66	3DI198N1	9684	4.86	-	-	-	-	9684	1219.82
66	3DI198N2	8944	3.35	-	-	-	-	8944	929.51
66	3DI198N3	9745	3.09	-	-	-	-	9745	767.66
33	3D1299N1	133	0.01	-	-	133	3.50	133	1.73
33	3D1299N2	131	0.03	-	-	131	1128.17	131	3.94
33	3D1299N3	131	0.02	131	1.98	131	580.97	131	3.31
66	3D1198N1	286	0.15	-	-	-	-	286	383.95
66	3D1198N2	286	0.16	-	-	-	-	286	341.05
66	3D1198N3	282	0.23	-	-	-	-	282	329.67

This result indicates that AMBS is able to run on every instance, and obtain a solution with high quality, while AMGO and the pure beam search are not able to deal with a number of instances. The running time of LSGA is quite short with high quality solution. This is because LSGA is an iterative algorithm and the instances in this dataset are easy to solve. If the instance is hard to solve, like the large instance in Balas and Saltzman Dataset, the quality of solutions of LSGA might not be that high.

5 Conclusion

In this paper, we propose a new heuristic named Approximate Muscle guided Beam Search (AMBS) for AP3 problem. This algorithm combines the approximate muscle and beam search. In this way, AMBS can achieve a good trade-off between the solution quality and the running time. Experimental results indicate that the new algorithm is able to obtain solutions with competitive quality, even on large instances.

Acknowledgement. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant DUT13RC(3)53, in part by the New Century Excellent Talents in University under Grant NCET-13-0073, in part by China Postdoctoral Science Foundation under Grant 2014M551083, and in part by the National Natural Science Foundation of China under Grant 61175062 and Grant 61370144.

References

1. Pierskalla, W.P.: The tri-substitution method for the three-dimensional assignment problem. CORS Journal 5, 71–81 (1967)
2. Pierskalla, W.P.: Letter to the Editor—The Multidimensional Assignment Problem. Operations Research 16, 422–431 (1968)
3. Frieze, A.M., Yadegar, J.: An Algorithm for Solving 3-Dimensional Assignment Problems with Application to Scheduling a Teaching Practice. The Journal of the Operational Research Society 32, 989–995 (1981)
4. Crama, Y., Kolen, A.W.J., Oerlemans, A.G., Spieksma, F.C.R.: Throughput rate optimization in the automated assembly of printed circuit boards. Ann. Oper. Res. 26, 455–480 (1991)
5. Balas, E., Saltzman, M.J.: An Algorithm for the Three-Index Assignment Problem. Operations Research 39, 150–161 (1991)
6. Crama, Y., Spieksma, F.C.R.: Approximation algorithms for three-dimensional assignment problems with triangle inequalities. European Journal of Operational Research 60, 273–279 (1992)
7. Burkard, R.E., Rudolf, R., Woeginger, G.J.: Three-dimensional axial assignment problems with decomposable cost coefficients. Discrete Applied Mathematics 65, 123–139 (1996)
8. Pardalos, P.M., Pitsoulis, L.S.: Nonlinear assignment problems: Algorithms and applications. Springer (2000)
9. Voss, S.: Heuristics for Nonlinear Assignment Problems. In: Pardalos, P., Pitsoulis, L. (eds.) Nonlinear Assignment Problems, vol. 7, pp. 175–215. Springer, US (2000)

10. Aiex, R.M., Resende, M.G.C., Pardalos, P.M., Toraldo, G.: GRASP with Path Relinking for Three-Index Assignment. INFORMS J. on Computing 17, 224–247 (2005)
11. Huang, G., Lim, A.: A hybrid genetic algorithm for the Three-Index Assignment Problem. European Journal of Operational Research 172, 249–257 (2006)
12. Jiang, H., Xuan, J., Zhang, X.: An approximate muscle guided global optimization algorithm for the Three-Index Assignment Problem. In: IEEE Congress on Evolutionary Computation, CEC 2008 (IEEE World Congress on Computational Intelligence), pp. 2404–2410 (2008)
13. Jiang, H., Chen, Y.: An efficient algorithm for generalized minimum spanning tree problem. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 217–224. ACM, Portland (2010)
14. Cazenave, T.: Monte Carlo Beam Search. IEEE Transactions on Computational Intelligence and AI in Games 4, 68–72 (2012)
15. López-Ibáñez, M., Blum, C.: Beam-ACO for the travelling salesman problem with time windows. Computers & Operations Research 37, 1570–1583 (2010)
16. Kim, B.-J., Hightower, W.L., Hahn, P.M., Zhu, Y.-R., Sun, L.: Lower bounds for the axial three-index assignment problem. European Journal of Operational Research 202, 654–668 (2010)