Phased Genetic Programming for Application to the Traveling Salesman Problem

Darren M. Chitty
Faculty of Environment, Science and Economy
University of Exeter, UK
darrenchitty@googlemail.com

Ed Keedwell
Faculty of Environment, Science and Economy
University of Exeter, UK
E.C.Keedwell@exeter.ac.uk

ABSTRACT

The Traveling Salesman Problem (TSP) is a difficult permutation-based optimisation problem typically solved using heuristics or meta-heuristics which search the solution problem space. An alternative is to find sets of manipulations to a solution which lead to optimality. Hyper-heuristics search this space applying heuristics sequentially, similar to a program. Genetic Programming (GP) evolves programs typically for classification or regression problems. This paper hypothesizes that GP can be used to evolve heuristic programs to directly solve the TSP. However, evolving a full program to solve the TSP is likely difficult due to required length and complexity. Consequently, a phased GP method is proposed whereby after a phase of generations the best program is saved and executed. The subsequent generation phase restarts operating on this saved program output. A full program is evolved piecemeal. Experiments demonstrate that whilst pure GP cannot solve TSP instances when using simple operators, Phased-GP can obtain solutions within 4% of optimal for TSPs of several hundred cities. Moreover, Phased-GP operates up to nine times faster than pure GP.

CCS CONCEPTS

• **Computing methodologies → Genetic programming**.

KEYWORDS

Genetic Programming, Traveling Salesman Problem

1 INTRODUCTION

The Traveling Salesman (TSP) is a well studied optimisation problem whereby the goal is to find the order in which to visit cities once only such that the total distance travelled is minimised. The symmetric TSP is represented as a complete weighted graph $G = (V, E, d)$ where $V = \{1, 2, .., n\}$ is a set of vertices defining cities and $E = \{(i, j)|(i, j) \in V \times V\}$ the edges with distance d between pairs of cities such that $d_{ij} = d_{ji}$. The objective is to find a Hamiltonian cycle in G of minimal length. The TSP is considered an $\mathcal{NP}$-Hard problem due to large feasible solution space.

Exact methods can solve small TSP instances but do not scale to larger problems. Consequently, meta-heuristic methods which do not guarantee an optimal result but a near optimal solution are popular methods. Genetic Algorithms (GAs) [9] are the most well known method used for solving TSP instances using the principles of Darwinian evolution to successively improve a population of candidate solutions where chromosome consists of the set of cities in the order to be visited. An alternative meta-heuristic is based upon the foraging behavior of ants. Ant Colony Optimisation (ACO) [7] applied to the TSP involves simulated ants traversing graph G visiting each vertex depositing pheromone defined by solution quality on edges taken. Ants probabilistically decide vertices to visit next using this pheromone level upon the edges of graph G and heuristic information of edge length. Meta-heuristics can be improved by computationally expensive local search methods such as 3-opt [12]). Indeed, local search is key to the state of the art heuristic Chained Lin-Kernighan for solving the TSP [2].

An alternative search is to find an optimal set of manipulations or operations to a given solution that will derive the optimal solution. Hyper-heuristics [5] are a method used to search this problem space [4]. Hyper-heuristics do not use problem domain knowledge within their operation. Instead, low level heuristics are utilised ranging from methods that are simple such as swapping two vertices in a solution to deploying a full local search methodology such as 2-opt. Advanced hyper-heuristic methods search for sequences of heuristics to be applied in a single iteration such as the Selection-Based Sequence Hyper-Heuristic (SSHH) [10].

However, a set of heuristics to be applied sequentially can also be considered as a *program* whereby each program line is a heuristic operation. A well known evolutionary methodology used to automatically generate programs is Genetic Programming (GP) [11]. GP uses the principles of Darwinian evolution namely natural selection, crossover and mutation, to successively improve a population of candidate programs. Typically, GP is applied to classification or symbolic regression problem. GP is not directly applied to optimisation problems such as the TSP. However, given the hyper-heuristics methodology of finding heuristic operations being similar to a program, GP is likely equally applicable.

GP is not commonly used to solve combinatorial problems such as the TSP directly. However, there are some exceptions for instance Dimopoulus and Zalzaha [6] used GP to evolve dispatching rules for the one-machine tardiness optimisation problem. Tay and Ho [16] also use GP to evolve dispatching rules for the flexible job-shop problem with the evolved rules representing a heuristic or rule of thumb. Within civil engineering Soh and Yang [15] use GP to evolve optimised structures in terms of size and geometry. However, although GP is not directly used for combinatorial optimisation it is used within a hyper-heuristic framework in a *generative* form whereby the goal is to synthesize a novel heuristic method that can be used to solve a problem such as the TSP within hyper-heuristic framework. For instance, Ryser-Welch et al. [14] use a cartisian form of GP to evolve TSP solvers that use advanced operators such as 3-opt local search. Duflo et al. [8] use GP to generate heuristics to solve TSP instances with improved results over nearest neighbour. In an alternative context, Nguyen et al. [13] use GP

ISBN 979-8-4007-0120-7/23/07
https://doi.org/10.1145/3583133.3590673

to decide heuristic selection within a hyper-heuristic framework to solve combinatorial optimisation problems.

2 APPLYING GP TO THE TSP

To directly apply GP to optimisation problems such as the TSP requires evolving a set of operations to an initial solution which will result in the optimal solution. Given a TSP instance and a random solution to the order of the cities to be visited it can be considered that a program exists that can reorder this to the optimal solution. Moreover, given a simple swap operation and an *n* city TSP it is conceivable there exists a program of swaps of *n* or less steps.

GP is commonly used in a tree-based form with lower branches providing inputs to higher levels but for application to the TSP this format is not applicable, a sequential sequence of operations is required. Essentially, a form of traditional program whereby each line constitutes an operation to be performed. A variant of GP known as Linear GP (LGP) [3] generates programs of this type.

Algorithm 1 Exemplar GP Program to Solve a TSP Instance

```
1: $S'$ = initial global starting solution $S$
2: $S'$ = SWAP(3,8)        {swap city at position 3 with city at position 8}
3: $S'$ = INSERT(7,2)      {insert city at position 7 into position 2}
4: $S'$ = INSERT(9,1)      {insert city at position 9 into position 1}
5: $S'$ = INVERT(3,6)      {invert cities between positions 3 and 6}
6: $S'$ = SWAP(4,5)        {swap city at position 4 with city at position 5}
7: $S'$ = INVERT(1,3)      {invert cities between positions 1 and 3}
8: $S'$ = INVERT(4,7)      {invert cities between positions 4 and 7}
```

With LGP a program of operations is evolved whereby the first element of each line refers to which operation to perform and the following parts the parameters to this operation. For applying LGP to the TSP using a simple swap operation these parameters would be constant values referring to positions within the solution, 1 to the number of cities. Algorithm 1 provides an example of a program that could be evolved by LGP for application to the TSP. LGP applied to the TSP evolves programs with the fitness measured by applying the program operations to the starting solution and assessing the result in terms of distance travelled.

2.1 Phased-GP

However, it could be considered that finding an optimal program to solve the TSP is more difficult than using a meta-heuristic. This is due to operations and their associated parameters needing to be evolved. If the operations necessary is equal to the number of cities and each has two parameters the solution space is three times larger. Moreover, long programs will be required. Consequently, a form of hill-climbing or *phased* approach is proposed for use within the LGP evolutionary process. Instead of evolving a complete large program that generates the optimal result from an initial random solution, a program can be evolved in stages or *phases*. Consider that at the evolutionary early stage small programs have evolved that improve upon the initial solution but not to optimality. Rather than attempting to grow these small programs into larger programs, the best improvement could be *locked in*. With the TSP, this program can be applied to the current solution to improve it and this becomes the new solution for subsequent evolved programs to be applied to. The next stage of program evolution operates on the output of this *saved* program. This methodology repeats until the conclusion of the evolutionary process. This is in effect hill-climbing to consistently improved solutions. This process can be termed as a *phased* approach to evolving a good program. Repeatedly doing this allows a program to be evolved *piecemeal*, each saved program can be output to provide the entire program that can optimise the initial solution.

Consider, the exemplar program in Algorithm 1 if evolved in phases. The code within lines 2-4 is the best program evolved in phase 1. At this point the program is *saved* and all subsequent programs operate on the output of this program. This is achieved by updating the initial solution S with current solution S'. Program lines 5-6 constitute the best program evolved by building upon the output program from phase 1. This too is *saved* and solution S updated solution S' and phase 3 then evolves the final 2 lines. The best programs from each phase constitute the complete solution.

Algorithm 2 Phased-GP Applied to the TSP

```
1: $S$ = current solution to TSP instance generated randomly
2: $S'$ = new solution
3: $S_{best}$ = best solution
4: $P_{best}$ = best program
5: while generation less than max generations do
6:     $P$ = generate population of random programs
7:     for number of phase generations do
8:         increment generation
9:         for each program $p$ in $P$ do
10:            $S'$ = executed program $p$ on $S$
11:            if $S'$ better than $S_{best}$ then
12:                $S_{best}$=$S'$
13:                $P_{best}$=$p$
14:            end if
15:        end for
16:        $P$ = generate new program population via selection, crossover, mutation
17:    end for
18:    if $S_{best}$ better than $S$ then
19:        $S$ = $S_{best}$
20:    end if
21:    reset $S_{best}$ and $P_{best}$
22: end while
```

The phased variant of GP (Phased-GP) is shown in Algorithm 2 whereby phased aspect is the additional inner loop on line 7. This loop evolves a program on the current solution and at the end of this loop the best found program is used to update the current solution essentially saving the program. Note that a new population of random solutions is generated at the end of each phase and the evolutionary process begins again. If no program has been evolved in the population that improves upon the current solution then this becomes in effect a restart. As optimality of the solution is approached this will occur frequently.

3 EXPERIMENTAL RESULTS

To measure the effectiveness of LGP and Phased-GP applied to the TSP they will be tested using six instances from the TSPLIB library described in Table 1. The GP approaches will use the parameters described in Table 2. A large degree of evolution is used since the objective is optimisation rather than generalisation. A high mutation rate is used since using GP to optimise operators and parameters, the introduction of new values into the population is necessary.

Table 1: TSP instances utilised

TSP Instance	Number of Cities	Optimal Solution	TSP Instance	Number of Cities	Optimal Solution
bier127	127	118282	ch130	130	6110
rat195	195	2323	d198	198	15780
a280	280	2579	pcb442	442	50778

Table 2: GP parameters

GP Parameters			
Population Size -	512	Max. Iterations -	100k
Crossover Prob. -	90%	Mutation Prob. -	33%
Tournament Size -	4	Terminal Set -	No. Cities
Operators -	Swap, Insert, Invert, 3-opt move		

Table 3: Relative errors, runtimes and program lengths when applying LGP to range of TSP instances.

TSP	Relative Error (%)			Runtime	Program
	Average	Best	Worst	in seconds	Length
bier127	137.18±23.98	103.91	192.43	190.62±10.19	76.99±4.83
ch130	257.90±43.13	194.16	357.60	222.07±16.19	90.16±6.94
rat195	316.10±42.45	258.05	459.63	284.13±32.39	119.41±7.75
d198	298.84±35.54	236.36	379.05	283.08±25.67	113.81±10.28
a280	488.89±51.19	402.57	629.61	435.54±35.55	170.62±9.98
pcb442	703.41±56.23	594.57	821.81	824.46±107.51	278.88±15.00

Four TSP operations are used with GP, swap, insertion, inversion and a single 3-opt move. The first three require two parameters, the fourth operator four. Therefore, chromosomes use an arity of five and each operator uses only the parameters it requires. Single point crossover is used ensuring that child chromosome arity remains constant. Three mutation operators are utilised with uniform probability, crossover, addition and modification. Crossover mutation performs 2-point crossover with a randomly generated chromosome. Addition mutation adds up to five random operators. Modification crossover makes up to five random changes to operators or parameters. A synchronous parallel implementation of GP is used with an eight core AMD Ryzen 2700 processor. Experiments are conducted over 25 random seeded execution runs.

To begin pure Linear GP will be applied to the TSP instances without any phases. These results shown in Table 3 are poor for even the smallest of the TSP instances. Clearly, GP when used to evolve a complete program of operations from a random initial solution cannot find programs that get close to optimality. A key reason is the size of the programs, a program of swaps or inserts less than the number of cities will exist that results in optimality. Program lengths are over half this length but have poor solution quality. Bloat has occurred with sequences evolving that do not impact the fitness [1]. For solving the TSP this can be a swap operation followed by another undoing this swap. Also, with long programs, a high degree of correct parameters need to be established.

To test the opposite approach of evolution, a null hypothesis will be tested whereby entirely random programs are generated at each generation. If the best population program improves upon the current solution then this program updates this solution as described for Phased-GP. The next population of randomly generated programs will then modify this new best solution. Clearly, no evolutionary process occurs, in effect simple hill-climbing. Table 4 shows the null hypothesis results whereby in complete contrast to using LGP, solutions are derived on average within less than 10% of optimality and the best solutions within 6%. Note the short

Table 4: Relative errors, runtimes and program lengths applying null hypothesis of no evolution to range of TSPs.

TSP	Relative Error (%)			Runtime	Program
	Average	Best	Worst	in seconds	Length
bier127	3.49±1.62	0.92	7.98	74.36±0.44	5.50±0.00
ch130	4.15±1.33	1.49	6.33	76.25±1.03	5.50±0.00
rat195	7.71±1.51	4.33	10.05	78.39±0.20	5.50±0.00
d198	2.79±1.18	0.95	4.58	79.55±0.27	5.50±0.00
a280	9.98±1.74	5.40	12.41	84.13±0.33	5.50±0.00
pcb442	8.52±1.72	5.57	11.57	94.70±0.36	5.50±0.00

Table 5: Relative errors, runtimes and program lengths applying Phased-GP to TSPs for range of phase generations

TSP	Phase Gens.	Relative Error (%)			Runtime	Program
		Average	Best	Worst	(secs)	Length
bier127	3	**3.25±1.56**	**0.25**	7.48	96.75 ±0.22	4.220±0.003
	5	4.04±1.91	1.68	7.96	91.11 ±5.25	3.190±0.003
	10	3.92±1.87	0.80	8.06	77.98 ±1.63	2.390±0.003
	20	4.06±2.42	1.57	8.84	72.02 ±1.28	2.000±0.003
	30	4.72±1.87	1.62	8.20	68.69 ±0.30	1.880±0.003
	50	4.53±2.16	0.88	9.06	66.13 ±0.37	1.810±0.007
	100	4.48±1.59	0.50	7.30	65.60 ±0.21	1.830±0.016
	500	5.17±1.99	1.89	9.69	69.31 ±0.83	3.330±0.203
ch130	3	**3.37±1.21**†	**0.51**	5.74	97.01 ±0.21	4.190±0.002
	5	3.44±1.33	1.37	6.68	86.28 ±0.26	3.160±0.001
	10	3.96±1.51	1.78	7.11	87.70 ±7.52	2.380±0.001
	20	3.87±1.23	1.54	6.73	72.20 ±1.33	1.990±0.002
	30	4.23±1.44	1.86	7.34	68.76 ±0.18	1.880±0.003
	50	4.76±1.68	1.95	8.18	68.61 ±1.54	1.800±0.007
	100	4.29±1.38	2.40	7.45	65.96 ±0.18	1.840±0.018
	500	5.10±1.54	2.74	8.87	68.63 ±0.79	3.450±0.262
rat195	3	6.33±1.50†	4.30	9.17	104.33 ±0.23	4.210±0.002
	5	6.29±1.31†	**4.18**	9.39	93.88 ±1.45	3.180±0.002
	10	**6.20±1.06**	4.42	8.93	82.53 ±0.25	2.390±0.001
	20	7.35±1.13	5.57	9.62	76.77 ±0.23	2.000±0.002
	30	7.44±1.38	5.18	10.64	74.01 ±0.21	1.890±0.005
	50	7.63±1.49	5.17	9.95	72.88 ±0.14	1.840±0.009
	100	8.06±1.50	5.31	11.54	71.63 ±0.19	1.930±0.028
	500	7.86±1.59	5.23	11.37	77.76 ±0.95	4.820±0.418
d198	3	**2.07±0.81**†	**0.74**	3.91	104.42 ±0.42	4.340±0.007
	5	2.20±0.76	0.95	3.67	94.30 ±0.41	3.270±0.009
	10	2.22±0.92	1.04	4.39	82.96 ±0.31	2.430±0.005
	20	2.54±1.02	0.87	4.66	76.61 ±0.57	2.020±0.003
	30	2.78±0.96	1.15	4.53	75.15 ±0.31	1.910±0.003
	50	3.16±0.82	1.56	4.64	73.55 ±0.32	1.850±0.007
	100	3.02±1.25	1.72	7.32	72.57 ±0.28	1.960±0.024
	500	3.34±1.17	0.87	5.23	79.08 ±0.68	5.010±0.281
a280	3	**7.84±1.74**†	**3.96**	10.96	113.36 ±0.28	4.190±0.002
	5	8.14±1.95†	4.90	11.39	100.62 ±0.34	3.160±0.002
	10	8.50±2.11†	4.81	13.72	90.53 ±0.31	2.380±0.001
	20	8.91±2.08	5.71	14.33	84.27 ±0.27	2.000±0.003
	30	9.41±2.02	6.17	13.64	81.78 ±0.29	1.900±0.006
	50	9.43±1.91	5.90	12.84	79.88 ±0.42	1.880±0.012
	100	10.20±2.38	4.77	14.49	80.21 ±0.26	2.060±0.032
	500	9.88±1.62	6.64	13.10	90.51 ±1.29	6.900±0.504
pcb442	3	**7.74±1.13**†	4.91	9.36	130.77 ±0.97	4.170±0.001
	5	7.81±1.00†	5.93	10.18	116.89 ±0.51	3.150±0.001
	10	7.77±1.52†	**3.80**	10.51	104.99 ±0.52	2.370±0.001
	20	8.40±1.36	5.05	10.16	98.00 ±0.63	2.010±0.002
	30	8.36±1.45	5.23	11.84	96.18 ±0.33	1.930±0.007
	50	8.24±1.59	5.53	12.58	94.32 ±0.59	1.960±0.014
	100	8.31±1.98	5.44	11.59	94.61 ±0.51	2.340±0.041
	500	9.06±1.19	6.96	12.03	120.40 ±2.56	12.510±0.719

†Statistically significant improvement of Phased-GP over null hypothesis with a $p < 0.05$ t-test, two-sided significance level and 24 degrees of freedom

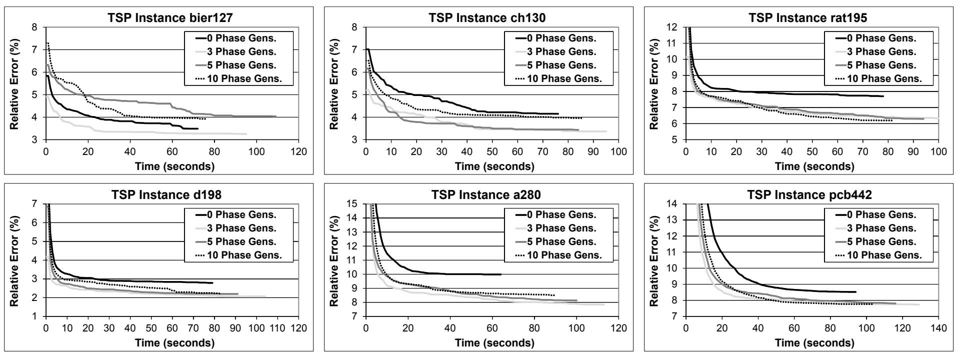

Figure 1: Average optimisation convergence rates over time for Phased-GP vs. null hypothesis

program lengths with only programs of up to 10 instructions generated also leading to a significant speedup over LGP. This demonstrates the advantage of hill climbing towards better solutions.

The results from Phased-GP with a range of generations between phases are shown in Table 5. Observe that in contrast to the null hypothesis of no evolution improved results are achieved when using a small degree of evolution within phases. Approximately 3-20 generations is most beneficial, frequent saving of small programs that improve the current solution, hill-climbing, is necessary to solve the TSP with GP. As evolution within phases increases results degrade in quality. Note the average program lengths decrease as evolution increases until a relatively high degree of several hundred generations. Given the available operators used with Phased-GP always modify a solution a long program will likely result in a poorer solution. Indeed, only short programs are likely to improve upon a near optimal solution. This acts as a *natural* parsimony pressure to reduce program length. However, this also leaves little genetic material for GP to utilise. Figure 1 shows the convergence rates of Phased-GP vs. the null hypothesis whereby a low degree of evolution within phases has the best convergence rate. Phased-GP is also up to 9x faster than LGP with superior results.

4 CONCLUSIONS

This paper has considered the novel use of Genetic Programming (GP) to directly solve the Traveling Salesman Problem (TSP). It was hypothesised that standard Linear GP would be unable to evolve programs that could solve even small TSP instances thus a Phased-GP approach was proposed which periodically saves the best program and *locks* this in. A new program is evolved on the resulting solution from this program and so forth. Programs could be evolved in piecemeal, a phased approach to GP (Phased-GP).

Experiments demonstrated Linear GP was incapable of evolving good solutions to TSP instances but Phased-GP demonstrated significant improvements by constructing programs in parts enabling a form of hill-climbing. Only a minimal degree of evolution of 3-20 generations within phases before saving best programs and restarting is beneficial achieving solutions within 4% of optimal. Note though that Phased-GP used simple operators and methods such as full 3-opt would improve results. Indeed, Phased-GP shows promise for application to combinatorial problems in general.

ACKNOWLEDGMENTS

Supported by Innovate UK [grant no. 10007532] and City Science.

REFERENCES

[1] Peter John Angeline. 1994. Genetic programming and emergent intelligence. *Advances in genetic programming* 1 (1994), 75–98.
[2] David Applegate, William Cook, and André Rohe. 2003. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing* 15, 1 (2003), 82–92.
[3] Markus Brameier and Wolfgang Banzhaf. 2001. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation* 5, 1 (2001), 17–26.
[4] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society* 64, 12 (2013), 1695–1724.
[5] Peter Cowling, Graham Kendall, and Eric Soubeiga. 2000. A hyperheuristic approach to scheduling a sales summit. In *International conference on the practice and theory of automated timetabling*. Springer, 176–190.
[6] Christos Dimopoulos and Ali MS Zalzala. 2001. Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in engineering software* 32, 6 (2001), 489–498.
[7] Marco Dorigo and Luca Maria Gambardella. 1997. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation* 1, 1 (1997), 53–66.
[8] Gabriel Duflo, Emmanuel Kieffer, Matthias R Brust, Grégoire Danoy, and Pascal Bouvry. 2019. A GP hyper-heuristic approach for generating TSP heuristics. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 521–529.
[9] John H Holland. 1975. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press.
[10] Ahmed Kheiri and Ed Keedwell. 2017. A hidden markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems. *Evolutionary computation* 25, 3 (2017), 473–501.
[11] John R. Koza. 1992. Genetic Programming.
[12] Shen Lin. 1965. Computer solutions of the traveling salesman problem. *Bell System Technical Journal* 44, 10 (1965), 2245–2269.
[13] Su Nguyen, Mengjie Zhang, and Mark Johnston. 2011. A genetic programming based hyper-heuristic approach for combinatorial optimisation. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 1299–1306.
[14] Patricia Ryser-Welch, Julian F Miller, Jerry Swan, and Martin A Trefzer. 2016. Iterative Cartesian genetic programming: creating general algorithms for solving travelling salesman problems. In *Genetic Programming: 19th European Conference, EuroGP 2016, Porto, Portugal, March 30-April 1, 2016, Proceedings 19*. Springer, 294–310.
[15] Chee Kiong Soh and Yaowen Yang. 2000. Genetic programming-based approach for structural optimization. *Journal of Computing in Civil Engineering* 14, 1 (2000), 31–37.
[16] Joc Cing Tay and Nhu Binh Ho. 2008. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering* 54, 3 (2008), 453–473.