HELMUT-SCHMIDT-UNIVERSITÄT
UNIVERSITÄT DER BUNDESWEHR HAMBURG
LEHRSTUHL FÜR BETRIEBSWIRTSCHAFTSLEHRE,
INSBES. LOGISTIK-MANAGEMENT
Prof. Dr. M. J. Geiger

Arbeitspapier / Research Report
RR-10-03-01 · March 2010 · ISSN 2192-0826

New Instances for the Single Machine Total Weighted Tardiness Problem

Martin Josef Geiger

Contents

Abstract

Previous research in the single machine total weighted tardiness problem (SMTWTP) has led to the proposition of effective local search strategies. At least existing benchmark instances from the literature do not pose a challenge for state-of-the-art algorithms.

This paper describes the proposition of two classes of novel instances for the single machine total weighted tardiness problem. In response to preceding research, they are larger, thus harder to search by local search algorithms. Besides, they are computed w. r. t. control parameters that lead to comparable difficult data sets.

In addition to providing novel instances, we report best known results, which have been computed by a Variable Neighborhood Descent algorithm.

1 Problem description

In the SMTWTP, a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ needs to be processed on a single machine. Each job J_j consists of a single operation only, involving a processing time $p_j > 0 \ \forall j = 1, \ldots, n$. The relative importance of the jobs is expressed via a nonnegative weight $w_j > 0 \ \forall j = 1, \ldots, n$. Processing on the machine is only possible for a single job at a time, excluding parallel processing of jobs. Each job J_j is supposed to be finished before its due date D_j. If this is not the case, a tardiness T_j occurs, measured as $T_j = \max\{s_j + p_j - D_j; 0\}$, where s_j denotes the starting time of job j. The overall objective of the problem is to find a feasible schedule x minimizing the total weighted tardiness TWT, i. e. min $TWT = \sum_{j=1}^{n} w_j T_j$.

A schedule for a particular problem can be interpreted as a vector of starting times of the jobs, $x = (s_1, \ldots, s_n)$. We assume that processing starts at time 0, thus $s_j \geqslant 0 \ \forall j = 1, \ldots, n$. A possible overlapping of jobs on the machine is avoided by the formulation of disjunctive side constraints: $s_j \geqslant s_k + p_k \ \dot{\vee} \ s_j + p_j \leqslant s_k \ \forall j, k = 1, \ldots, n, j \neq k$.

As the objective function of the single machine total weighted tardiness problem is a *regular* function [1], it is known that at least one active schedule exists which is also optimal. A schedule is called *active* if, for a given sequence of jobs, all operations are started as early as possible, thus avoiding all unnecessary in-between waiting times (delays). The problem of finding an optimal schedule may therefore be reduced to the problem of finding an optimal sequence of jobs. A given sequence is represented by a permutation $\pi = \{\pi_1, \ldots, \pi_n\}$ of the job indices. Each element π_i in π stores the index of the job which is to be processed as the ith job in the processing sequence. The permutation of indices is then 'decoded' into a schedule by assigning $s_{\pi_1} = 0$ and computing the values of $s_{\pi_i} = s_{\pi_{i-1}} + p_{\pi_{i-1}} \ \forall i = 2, \ldots, n$.
Obviously, this leads to an active schedule without any waiting times between jobs.

2 Other (old) benchmark data from the literature

Optimization approaches for the SMTWTP are commonly verified using the benchmark instances of [4]. The authors presented 375 data sets with varying characteristics. For values of $n = 40$, $n = 50$, and $n = 100$, 125 instances are each proposed. The computation of the processing times p_j is randomly drawn from a uniform distribution $[1, 100]$, while the weights are taken from $[1, 10]$. Depending on the relative range of due dates RDD and the average tardiness factor TF, the due dates are randomly computed as integer values within $\left[P\left(1 - TF - \frac{RDD}{2}\right), P\left(1 - TF + \frac{RDD}{2}\right)\right]$, where $P = \sum_{j=1}^{n} p_j$. Five instances have been computed for each combination of RDD and TF: $RDD \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$, $TF \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$.

All data sets are available in the OR-Library under `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`. For the ones with $n = 40$ and $n = 50$, optimal solutions are known, while for the ones of size $n = 100$, at least to our knowledge, only best-known solutions are reported in the literature, lacking their final proof of optimality. It should be noticed however, that the results for the larger data sets are commonly assumed to be optimal, as, despite rather active research in this area, there has not been any improvement of the best-known upper bounds within past ten years.

It has been pointed out that available benchmark instances are comparably easily solvable by local search [6]. A particularly successful neighborhood search technique for the SMTWTP is *Iterated Dynasearch* [3], which uses dynamic programming to determine an optimal series of moves to be executed simultaneously. More recently, several different metaheuristics have been developed for the SMTWTP, successfully solving benchmark instances from the scientific literature. Important work includes simple local search [9], Evolutionary Algorithms [4], Ant Colony Optimization [5, 10, 2], Iterated Local Search [6], and Simulated Annealing [11].

Reviewing previous research in the SMTWTP, we may conclude that the well-known benchmark instances of [4] do not present a challenge for state-of-the-art algorithms any longer. Future research should therefore be able to make use of harder data sets, and the following section describes the proposition of new ones.

3 New instances

3.1 Large instances

New instances have been proposed using the control parameters described in Section 2. We chose $n = 1000$, thus creating considerable larger data sets. In order to decrease the similarity of the jobs, we chose to draw the processing times p_j from uniform $[1, 1000]$, and w_j from uniform $[1, 100]$. TF has been set to $TF = 0.5$, and $RDD \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. The choice of $TF = 0.5$ is based on experimental results, which indicate that instances based on this value are relatively more difficult to solve [7].

Five instances have been computed for each combination of TF and RDD, leading to a total of 25 data sets. The following Table 1 gives an overview about the proposed instances and the control parameters used for their computation.

Table 1: Overview about the proposed large benchmark instances

TF	RDD	Instances
0.5	0.2	`wt_1000_1, ..., wt_1000_5`
0.5	0.4	`wt_1000_6, ..., wt_1000_10`
0.5	0.6	`wt_1000_11, ..., wt_1000_15`
0.5	0.8	`wt_1000_16, ..., wt_1000_20`
0.5	1.0	`wt_1000_21, ..., wt_1000_25`

3.2 'Prime' instances

In addition to the large instances presented above, novel data sets have been generated that employ prime numbers for the weights and processing times. By ensuring that each combination of weight and processing time values is unique for all jobs of an instance, symmetries, such as otherwise present, can be avoided.

The weight values w_j are randomly drawn from a set W of 100 prime numbers, starting with 101, and the processing times p_j are drawn from the set P of 1061 prime numbers between 1009 and 9973. Both sets are documented in appendix A.

Again, TF has been set to $TF = 0.5$, and $RDD \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. Contrary to the instances from section 3.1, n is here $n = 100$. The following table 2 gives an overview about the obtained data sets.

Table 2: Overview about the proposed 'prime' benchmark instances

TF	RDD	Instances
0.5	0.2	`wt_100_prime_1, ..., wt_100_prime_5`
0.5	0.4	`wt_100_prime_6, ..., wt_100_prime_10`
0.5	0.6	`wt_100_prime_11, ..., wt_100_prime_15`
0.5	0.8	`wt_100_prime_16, ..., wt_100_prime_20`
0.5	1.0	`wt_100_prime_21, ..., wt_100_prime_25`

4 Computation of upper bounds

4.1 Local search approach

We did use the Variable Neighborhood Descent algorithm as described in [7] to compute local optima for the above introduced benchmark instances. The algorithm is primarily based on Variable Neighborhood Search (VNS) [8]. Contrary to VNS however, a random restart is carried out once a solution is reached which is a local optimum with respect to all employed neighborhoods.

Three neighborhoods are implemented: Exchange EX, swopping the position of two jobs, and forward FSH and backward shift BSH, which shift the position of a single job.

1000 random restarts were done for the 'prime' instances, each time starting search from a random permutation of jobs. In case of the larger instances with $n = 1000$, computing a local optimum turns out to be very time consuming, taking several weeks/ months on an Intel Xeon X5550 processor, running at 2.67 GHz. Here, the obtained results are those of a single run only.

4.2 Results

The best results of all test runs are given in table 3. Clearly, the best value of the total weighted tardiness decreases with increasing RDD, which is to be expected and also observed for the old data sets of [4]. On the other hand, and contrary to several instances from [4], no instance appears to exist for which all jobs finish before their due date.

Table 3: Best known results for the new benchmark instances

Instance	best upper bound	instance	best upper bound
wt_1000_1	661,554,849	wt_100_prime_1	802,910,360
wt_1000_2	685,100,253	wt_100_prime_2	755,883,602
wt_1000_3	579,874,750	wt_100_prime_3	772,936,285
wt_1000_4	632,097,650	wt_100_prime_4	772,242,365
wt_1000_5	667,684,534	wt_100_prime_5	772,266,246
wt_1000_6	468,054,238	wt_100_prime_6	584,502,396
wt_1000_7	397,169,788	wt_100_prime_7	651,160,845
wt_1000_8	428,824,156	wt_100_prime_8	485,031,706
wt_1000_9	398,764,545	wt_100_prime_9	600,247,834
wt_1000_10	479,737,599	wt_100_prime_10	538,889,310
wt_1000_11	229,786,779	wt_100_prime_11	217,962,855
wt_1000_12	224,986,621	wt_100_prime_12	441,851,304
wt_1000_13	227,899,643	wt_100_prime_13	317,231,662
wt_1000_14	212,283,556	wt_100_prime_14	295,071,692
wt_1000_15	220,489,886	wt_100_prime_15	248,660,251
wt_1000_16	67,354,310	wt_100_prime_16	292,956,061
wt_1000_17	69,668,511	wt_100_prime_17	212,953,095
wt_1000_18	74,724,105	wt_100_prime_18	189,128,175
wt_1000_19	68,685,125	wt_100_prime_19	111,382,882
wt_1000_20	52,528,847	wt_100_prime_20	138,182,878
wt_1000_21	11,134,727	wt_100_prime_21	166,716,374
wt_1000_22	8,843,784	wt_100_prime_22	147,222,671
wt_1000_23	13,831,801	wt_100_prime_23	58,998,980
wt_1000_24	8,102,595	wt_100_prime_24	108,268,446
wt_1000_25	15,287,150	wt_100_prime_25	163,480,751

5 Conclusions

In this article, we introduced two new sets of benchmark instances for the single machine total weighted tardiness problem. The data sets have been formulated in response to previous research, which indicated that existing benchmarks may easily be solved with state-of-the-art algorithms. In contrast to these older data sets, our instances are larger, and considerable less symmetric.

First, and thus currently best-know results have been computed on the basis of Variable Neighborhood Search Descent, given reference values for further comparisons.

All benchmark instances may be obtained from `http://logistik.hsu-hh.de/SMTWTP`.

A Sets W and P for the 'prime' instances

W = {101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691}

P = {1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719, 2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187, 3191, 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343, 3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527, 3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719, 3727, 3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001, 4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091, 4093, 4099, 4111, 4127, 4129, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231, 4241, 4243, 4253, 4259, 4261, 4271, 4273, 4283, 4289, 4297, 4327, 4337, 4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409, 4421, 4423, 4441, 4447, 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523, 4547, 4549, 4561, 4567, 4583, 4591, 4597, 4603, 4621, 4637, 4639, 4643, 4649, 4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721, 4723, 4729, 4733, 4751, 4759, 4783, 4787, 4789, 4793, 4799, 4801, 4813, 4817, 4831, 4861, 4871, 4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937, 4943, 4951, 4957, 4967, 4969, 4973, 4987, 4993, 4999, 5003, 5009, 5011, 5021, 5023, 5039, 5051, 5059, 5077, 5081, 5087, 5099, 5101, 5107, 5113, 5119, 5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227, 5231, 5233, 5237, 5261, 5273, 5279, 5281, 5297, 5303, 5309, 5323, 5333, 5347, 5351, 5381, 5387, 5393, 5399, 5407, 5413, 5417, 5419, 5431, 5437, 5441, 5443, 5449, 5471, 5477, 5479, 5483, 5501, 5503, 5507, 5519, 5521, 5527, 5531, 5557, 5563, 5569, 5573, 5581, 5591, 5623, 5639, 5641, 5647, 5651, 5653, 5657, 5659, 5669, 5683, 5689, 5693, 5701, 5711, 5717, 5737, 5741, 5743, 5749, 5779, 5783, 5791, 5801, 5807, 5813, 5821, 5827, 5839, 5843, 5849, 5851, 5857, 5861, 5867, 5869, 5879, 5881, 5897, 5903, 5923, 5927, 5939, 5953, 5981, 5987, 6007, 6011, 6029, 6037, 6043, 6047, 6053, 6067, 6073, 6079, 6089, 6091, 6101, 6113, 6121, 6131, 6133, 6143, 6151, 6163, 6173, 6197, 6199, 6203, 6211, 6217, 6221, 6229, 6247, 6257, 6263, 6269, 6271, 6277, 6287, 6299, 6301, 6311, 6317, 6323, 6329, 6337, 6343, 6353, 6359, 6361, 6367, 6373, 6379, 6389, 6397, 6421, 6427, 6449, 6451, 6469, 6473, 6481, 6491, 6521, 6529, 6547, 6551, 6553, 6563, 6569, 6571, 6577, 6581, 6599, 6607, 6619, 6637, 6653, 6659, 6661, 6673, 6679, 6689, 6691, 6701, 6703, 6709, 6719, 6733, 6737, 6761, 6763, 6779, 6781, 6791, 6793, 6803, 6823, 6827, 6829, 6833, 6841, 6857, 6863, 6869, 6871, 6883, 6899, 6907, 6911, 6917, 6947, 6949, 6959, 6961, 6967, 6971, 6977, 6983, 6991, 6997, 7001, 7013, 7019, 7027, 7039, 7043, 7057, 7069, 7079, 7103, 7109, 7121, 7127, 7129, 7151, 7159, 7177, 7187, 7193, 7207, 7211, 7213, 7219, 7229, 7237, 7243, 7247, 7253, 7283, 7297, 7307, 7309, 7321, 7331, 7333, 7349, 7351, 7369, 7393, 7411, 7417, 7433, 7451, 7457, 7459, 7477, 7481, 7487, 7489, 7499, 7507, 7517, 7523, 7529, 7537, 7541, 7547, 7549, 7559, 7561, 7573, 7577, 7583, 7589, 7591, 7603, 7607, 7621, 7639, 7643, 7649, 7669, 7673, 7681, 7687, 7691, 7699, 7703, 7717, 7723, 7727, 7741, 7753, 7757, 7759, 7789, 7793, 7817, 7823, 7829, 7841, 7853, 7867, 7873, 7877, 7879, 7883, 7901, 7907, 7919, 7927, 7933, 7937, 7949, 7951, 7963, 7993, 8009, 8011, 8017, 8039, 8053, 8059, 8069, 8081, 8087, 8089, 8093, 8101, 8111, 8117, 8123, 8147, 8161, 8167, 8171, 8179, 8191, 8209, 8219, 8221, 8231, 8233, 8237, 8243, 8263, 8269, 8273, 8287, 8291, 8293, 8297, 8311, 8317, 8329, 8353, 8363, 8369, 8377, 8387, 8389, 8419, 8423, 8429, 8431, 8443, 8447, 8461, 8467, 8501, 8513, 8521, 8527, 8537, 8539, 8543, 8563, 8573, 8581, 8597, 8599, 8609, 8623, 8627, 8629, 8641, 8647, 8663, 8669, 8677, 8681, 8689, 8693, 8699, 8707, 8713, 8719, 8731, 8737, 8741, 8747, 8753, 8761, 8779, 8783, 8803, 8807, 8819, 8821, 8831, 8837, 8839, 8849, 8861, 8863, 8867, 8887, 8893, 8923, 8929, 8933, 8941, 8951, 8963, 8969, 8971, 8999, 9001, 9007, 9011, 9013, 9029, 9041, 9043, 9049, 9059, 9067, 9091, 9103, 9109, 9127, 9133, 9137, 9151, 9157, 9161, 9173, 9181, 9187, 9199, 9203, 9209, 9221, 9227, 9239, 9241, 9257, 9277, 9281, 9283, 9293, 9311, 9319, 9323, 9337, 9341, 9343, 9349, 9371, 9377, 9391, 9397, 9403, 9413, 9419, 9421, 9431, 9433, 9437, 9439, 9461, 9463, 9467, 9473, 9479, 9491, 9497, 9511, 9521, 9533, 9539, 9547, 9551, 9587, 9601, 9613, 9619, 9623, 9629, 9631, 9643, 9649, 9661, 9677, 9679, 9689, 9697, 9719, 9721, 9733, 9739, 9743, 9749, 9767, 9769, 9781, 9787, 9791, 9803, 9811, 9817, 9829, 9833, 9839, 9851, 9857, 9859, 9871, 9883, 9887, 9901, 9907, 9923, 9929, 9931, 9941, 9949, 9967, 9973}

References

[1] Kenneth R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley & Sons, New York, London, Sydney, Toronto, 1974.

[2] Andreas Bauer, Bernd Bullnheimer, Richard F. Hartl, and Christine Strauss. An ant colony optimization approach for the single machine total tardiness problem. In *Proceedings of the 1999 Congress on Evolutionary Computation CEC99*, volume 2, pages 1445–1450, Piscataway, NJ, 1999. IEEE Service Center.

[3] Richard K. Congram, Chris N. Potts, and Steef L. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.

[4] H. A. J. Crauwels, C. N. Potts, and L. N. Van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 10(3):341–350, 1998.

[5] Matthijs den Besten, Thomas Stützle, and Marco Dorigo. Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature-PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 611–620, Berlin, Heidelberg, New York, 2000. Springer Verlag.

[6] Matthijs den Besten, Thomas Stützle, and Marco Dorigo. Design of iterated local search algorithms – an example application to the single machine total weighted tardiness problem. In E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, S. Cagnoni, E. Hart, G. R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 441–451, Berlin, Heidelberg, New York, 2001. Springer Verlag.

[7] Martin Josef Geiger. On heuristic search for the single machine total weighted tardiness problem – some theoretical insights and their empirical verification. *European Journal of Operational Research*, 207(3):1235–1243, 2010.

[8] Pierre Hansen and Nenad Mladenović. Developments of variable neighborhood search. In Celso C. Ribeiro and Pierre Hansen, editors, *Essays and Surveys in Metaheuristics*, chapter 19, pages 415–439. Kluwer Academic Publishers, Boston, Dordrecht, London, 2002.

[9] R. Maheswaran and S. G. Ponnambalan. An investigation on single machine total weighted tardiness scheduling problems. *The International Journal of Advanced Manufacturing Technology*, 22(3–4):243–248, 2003.

[10] Daniel Merkle and Martin Middendorf. An ant algorithm with a new pheromone evaluation rule for total tardiness problems. In S. Cagnoni, R. Poli, G. D. Smith, D. Corne, M. Oates, E. Hart, P. L. Lanzi, E. J. Willem, Y. Li, B. Paechter, and T. C. Fogarty, editors, *Real-World Applications of Evolutionary Computing*, volume 1903 of *Lecture Notes in Computer Science*, pages 287–296, Berlin, Heidelberg, New York, 2000. Springer Verlag.

[11] A. C. Nearchou. Solving the single machine total weighted tardiness scheduling problem using a hybrid simulated annealing algorithm. In *2nd IEEE International Conference on Industrial Informatics INDIN'04*, pages 513–516, Berlin, Germany, June 2004.