THE METHODOLOGY OF PARALLEL MEMETIC ALGORITHMS DESIGNING

Wojciech Bożejko
Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland
wojciech.bozejko@pwr.wroc.pl

Mieczysław Wodecki
University of Wrocław, Joliot-Curie 15, 50-383 Wrocław, Poland
mwd@ii.uni.wroc.pl

Keywords: Metaheuristics, Parallel computing, Evolutionary algorithm, Memetic algorithm.

Abstract: The paper presents the methodology of parallel algorithm designing based on the memetic approach (Lamarck and Baldwin evolution theory) making use of specific properties of the problem and distributed island model. This approach is presented on the example of the single machine scheduling problem with earliness/tardiness penalties.

1 INTRODUCTION

The memetic algorithm is an evolutionary approach based on the process of natural evolution adhering to the principles of natural selection, crossover and survival. The Lamarck's model (Michalewicz, 1994) of evolution is applied to intensify the optimization process. In each generation a certain part of the population is replaced by their local minima simulating a learning effect which can be succeeded by the next generation as a 'meme'. From the current population some subset is drawn. Each individual of this subset is a starting solution for the local optimization algorithm. Thus, there are five essential steps of the memetic algorithm:

1. *selection* – choosing some subset of individuals, so-called parents,
2. *crossover* – combining parts from pairs of parents to generate new ones,
3. *mutation* – transformation that creates a new individual by small changes applied to an existing one taken from the population,
4. *learning* – an individual is improved (e.g. by a local optimization),
5. *succession* – determining the next generation's population.

New individuals created by crossover or mutation replace all or a part of the old population. The process of evaluating fitness and creating a new population generation is repeated until a termination criterion is achieved.

Similar to the GA, following kinds of parallelization are usually applied to memetic algorithms MAs:

- global parallelization,
- independent runs,
- island model,
- diffusion model,

with similar properties as applied to the classic GA. Additionally, a local search procedure can be parallelized in MA. Such an approach is proposed by (Berger and Barkaoui, 2002) and applied to the Vehicle Routing Problem with Time Windows (VRPTW) by using a master-slave parallel approach. The master controls the memetic algorithm execution, synchronizes and handles parent selection while the slaves execute genetic operations together with local search in parallel. Parallel memetic algorithm was also considered by (Bradwell and Brown, 1999) (asynchronous MA) and (Tang et al., 2006) (MA based on population entropy).

Implementation of algorithms which are based on multithread multiple-walk searching of the solutions space are usually coarse-grained application, i.e. they require sparse communication and synchronization. These type of algorithms are easy to be applied in distributed calculation systems, as clusters which express beneficial efficiency-to-price ratio. Apart from speeding up the calculations, it is possible to improve

quality of obtained results. Search processes can be either independent or cooperative.

1.1 Independent Searching Threads

In this category we can distinguish two base approaches:

- Researching of the solution space by using multiple trajectories, which begin from different starting solutions (or different starting populations in the case of using population-based approaches). Searching threads can use either the same or different strategies, i.e. the same or different local search algorithms, the same or different parameters (tabu list length, population size, etc.). Trajectories can cross themselves in one or more places of the neighborhood graph.
- Parallel researching of subgraphs of a neighborhood graph obtained by decomposition of the problem into a few subproblems (for example fixing of some variables). Subgraphs of the neighborhood graph are searched concurrently without crossing search trajectories. We obtain partitioning of the neighborhood graph into disjoint subgraphs.

The first parallel implementation of the tabu search method based on multiple-walk searching of the solution space was proposed by Taillard for the quadric assignment problem (QAP) (Taillard, 1991) and the job shop problem (Taillard, 1994). The multiple-walk parallelization strategy based on independent searching threads is easy in implementation and one can obtain good values of the speedup under condition of proper decomposition of the solution space into searching threads (and their trajectories).

1.2 Cooperative Searching Threads

This model constitutes the most general and promising type of solution space searching strategy by using parallel metaheuristics, however it requires knowledge of solving problem specificity. 'Cooperative' means here the interchange of information – experience of searching history up to now. Specific information, which is characteristic for the problem and the method (i.e. the best solution found so far, elite solutions, the frequency of moves, tabu lists, backtrack-jump list, subpopulations and their sizes, etc.) has to be exchanged or broadcasted.

Information shared by search processes can be stored as global variables kept in the *shared memory* or as records in the local memory of the dedicated central processor which communicates with all other processors providing them with requested data. In the model, in which information gathered during moving along a trajectory is used to improve other trajectories, not only can one expect convergence of such a parallel algorithm, but also founding in the same time a better solution than the parallel algorithm without communication can take place. In such a case we can say that cooperative concurrent algorithms constitute a new class of algorithms in deed.

The first heuristic algorithm of this type was asynchronous parallel tabu search algorithm proposed by Crainic, Toulouse and Gendreau (Crainic et al., 1995). Packages such as ParSA (Kliewer et al., 1999) offer ready implementations of parallel simulated annealing algorithms based on cooperative searching threads. The interaction strategy is also very efficient in implementation of parallel genetic algorithms (in the sense of obtained solutions). There are plenty of ready libraries such as POOGAL (Bubak and Sowa, 1999). The majority of cooperative implementations of parallel genetic algorithm is based on the *migration island model*. Each process has its own subpopulation exchanging from time to time a number of individuals (usually the best – elite) with other processes (Bubak and Sowa (Bubak and Sowa, 1999), Crainic and Toulouse (Crainic and Toulouse, 1998)). Bożejko (Bożejko, 2010) proposed a parallel path-relinking metaheuristics based on the parallel scatter search algorithm.

1.3 The Problem

This paper aim is to present a parallel memetic approach on the instance of a strongly NP-hard scheduling problem. We additionally assume that the considered problem has *no idle* constraint (*TWET-no-idle* problem), which means that the machine works without stops. The problem of scheduling with earliness and tardiness (total weighted earliness/tardiness problem, *TWET*) is one of the most frequently considered in literature. In this problem each job from a set $\mathcal{J} = \{1,2,\ldots,n\}$ has to be processed, without interruption, on a machine, which can execute at most one job in each moment. By p_i we represent the execution time of a job $i \in \mathcal{J}$, and by e_i and d_i we mean an adequately demanded earliest and latest moment of the finishing processing of a job. If scheduling of jobs is established and C_i is the moment of finishing a job i, then we call $E_i = \max\{0, e_i - C_i\}$ an *earliness* and $T_i = \max\{0, C_i - d_i\}$ a *tardiness*. The expression $u_i E_i + w_i T$ is the *cost* of job execution, where u_i and w_i ($i \in \mathcal{J}$) are nonnegative coefficients of a goal function. The problem consists in minimizing a sum of costs of jobs, that is to find a job sequence $\pi^* \in \Phi_n$

such that for the goal function

$$F(\pi) = \sum_{i=1}^{n} (u_{\pi(i)} E_{\pi(i)} + w_{\pi(i)} T_{\pi(i)}), \quad \pi \in \Phi_n, \quad (1)$$

we have

$$F(\pi^*) = \min_{\pi \in \Phi_n} F(\pi). \quad (2)$$

This problem is represented by $1||\sum(u_i E_i + w_i T_i)$ in literature and it belongs to a strongly NP-hard class (if we assume $u_i = 0$, $i = 1,2, \ldots, n$, we will obtain a strongly NP-hard problem $1||\sum w_i T_i$ - Lenstra et al. (Lenstra et al., 1977)). Baker and Scudder (Baker and Scudder, 1990) proved, that there can be an idle time in an optimal solution (jobs need not to be processed directly one after another), that is $C_{\pi(i+1)} - p_{\pi(i+1)} \geq C_{\pi(i)}$, $i = 1,2,\ldots,n-1$. Solving the problem amounts to establishing a sequence of jobs and its starting times. Hoogeven and van de Velde (Hoogeveen and van de Velde, 1996) proposed an algorithm based on the branch and bound method. Because of exponentially growing computation time, this algorithm can be applied only to solve instances where the number of jobs is not greater than 20. Therefore, in practice almost always approximate algorithms are used. The best ones are based on artificial intelligence methods. Calculations are performed in two stages.

- Determining the scheduling of jobs (with no idle times).
- Establishing optimal starting times of jobs.

Bożejko and Wodecki (Bożejko and Wodecki, 2005) proposed a parallel coevolutionary algorithm for the considered problem.

1.3.1 Block Properties

For the *TWET-no-idle* problem, each schedule of jobs can be represented by permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of elements of the set of jobs $\mathcal{J}$. Let Φ_n denote the set of all such permutations. The total cost $\pi \in \Phi_n$ is $F(\pi) = \sum_{i=1}^{n} f_{\pi(i)}(C_{\pi(i)})$, where $C_{\pi(i)}$ is completed time of the job $\pi(i)$, $C_{\pi(i)} = \sum_{j=1}^{i} p_{\pi(j)}$. The job $\pi(i)$ is considered *early* if it is completed before its earliest moment of finishing ($C_{\pi(i)} < e_{\pi(i)}$), *on time* if $e_{\pi(i)} \leq C_{\pi(i)} \leq d_{\pi(i)}$, and *tardy* if the job is completed after its due date (i.e. $C_{\pi(i)} > d_{\pi(i)}$).

Each permutation $\pi \in \Phi_n$ is decomposed into subpermutations (subsequences of jobs) $B = (B^1, B^2, \ldots, B^v)$ called *blocks* in π, where:

1. $B^i = (\pi(a_i), \pi(a_i+1), \ldots, \pi(b_i-1), \pi(b_i))$, and $a_i = b_{i-1} + 1$, $1 \leq i \leq v$, $a_0 = 0$, $b_v = n$.
2. All the jobs $j \in B^i$ satisfy the following conditions:
   $e_j > C_{\pi(b_i)}$, (C1)
   $e_j \leq C_{\pi(b_{i-1})} + p_j$ and $d_j \geq C_{\pi(b_i)}$, (C2)
   $d_j < C_{\pi(b_{i-1})} + p_j$. (C3)
3. B_i are maximal subsequences of π in which all the jobs satisfy either Condition $C1$ or Condition $C2$ or Condition $C3$.

By definition, there exist three types of blocks implied by either C1 or C2 or C3. To distinguish them, we will use the *E-block*, *O-block* and *T-block* notions respectively. For any block Υ in a partition B of permutation $\pi \in \Phi_n$, let

$$F_\Upsilon(\pi) = \sum_{i \in \Upsilon} (u_i E_i + w_i T_i). \quad (3)$$

Therefore, the value of a goal function

$$F(\pi) = \sum_{i=1}^{n} (u_i E_i + w_i T_i) = \sum_{\Upsilon \in B} F_\Upsilon(\pi). \quad (4)$$

If Υ is a T-block, then every job inside is early. Therefore, an optimal sequence of the jobs within Υ of the permutation π (that is minimizing $F_\Upsilon(\pi)$) can be obtained, using the well-known Weighted Shortest Processing Time (*WSPT*) rule, proposed by Smith (Smith, 1956). The *WSPT* rule creates an optimal sequence of jobs in the non-increasing order of the ratios w_j/p_j. Similarly, if Υ is an E-block, than an optimal sequence of the jobs within can be obtained, using the Weighted Longest Processing Time (*WLPT*) rule which creates a sequence of jobs in the non-decreasing order of the ratios u_j/p_j. Partition B of the permutation π is *ordered*, if there are jobs in the *WSPT* sequence in any T-block, and if there are jobs in the *WLPT* sequence in any E-block.

Theorem 1 ((Bożejko et al., 2006)). *Let Υ be an ordered partition of a permutation $\pi \in \Phi_n$ to blocks. If $\beta \in \Phi_n$ and $F(\beta) < F(\pi)$, so at least one job of some block of π was moved before the first or after the last job of this block in the permutation β.*

Note that Theorem 1 provides the necessary condition to obtain a permutation β from π such, that $F(\beta) < F(\pi)$. Let $B = (B^1, B^2, \ldots, B^v)$ be an ordered partition of the permutation $\pi \in \Phi_n$ to blocks. If a job $\pi(j) \in B^i$ ($B^i \in B$), therefore moves which can improving goal function value consists in reordering a job $\pi(j)$ before the first or after the last job of this block. Let N_j^{bf} and N_j^{af} be sets of such moves ($N_j^{bf} = \emptyset$ for $j \in B^1$ and $N_j^{af} = \emptyset$ for $j \in B^v$). Therefore, the neighborhood of the permutation $\pi \in \Phi_n$,

$$N(\pi) = \bigcup_{j=1}^{n} N_j^{bf} \cup \bigcup_{j=1}^{n} N_j^{af}. \quad (5)$$

As computational experiments show, the neighborhood defined in (5) has a half smaller size than the neighborhood of all the insert moves.

2 MEMETIC ALGORITHM

All operations in a coevolutionary memetic algorithm (selection, crossover, local optimization and succession) are executed locally, on some subsets of the current population called *islands*. It is a strongly decentralized model of an evolutionary algorithm. There are independent evolution processes on each of the islands, and communication takes place sporadically. Exchanging individuals between islands secures diversity of populations and prevents fast imitating of an individual with a local minimum as its goal function. On each island a hybrid algorithm is applied, in which an evolutionary algorithm is used to determine the starting solutions for the local search algorithm. The outline of the standard memetic algorithm is presented on the Fig. 1.

```
Algorithm 1.  Memetic algorithm
Number of iteration k :=0;
P_0 ←initial population;
repeat
  P'_k ←Selection(P_k);
  P''_k ←Crossover(P'_k);
  P''_k ←Mutation(P''_k);
  A ←RandomSubSet(P''_k);
  P''_k ← P''_k ∪LocalMinimumSet(A);
  P_{k+1} ←Succession(P_k, P''_k)
  k := k + 1;
until some termination condition is satisfied;
```

Figure 1: Outline of the memetic algorithm.

3 PARALLEL MEMETIC ALGORITHM

The parallel algorithms based on the island model divide the population into a few subpopulations. Each of them is assigned to a different processor which performs a sequential memetic algorithm based on its own subpopulation. The crossover involves only individuals within the same population. Occasionally, the processor exchanges individuals through a migration operator. The main determinants of this model are: (1) size of the subpopulations, (2) topology of the connection network, (3) number of individuals to be exchanged, (4) frequency of exchanging. The island model of parallel memetic algorithm is characterized by a significant reduction of the communication time, compared to the global model (with distributed computations of the fitness function only). As shared memory is not required, this model is also more flexible.

Below, a parallel memetic algorithm is proposed. The algorithm is based on the island model of parallelism (see Bożejko and Wodecki (Bożejko and Wodecki, 2006)). We have adapted the MSXF (Multi – Step Crossover Fusion) operator which is used to extend the process of researching for better solutions of the problem. Originally, a MSXF has been described by Reeves and Yamada (Reeves and Yamada, 1998). Its idea is based on local search, starting from one of the parent solutions, to find a new good solution where the other parent is used as a reference point. Here we propose to use block properties defined in the Section 1.3.1 to make the search process more effective – prevent changes inside the block (which are unprofitable from the fitness function's point of view). Such a proceeding is consistent with an idea of not making unprofitable changes between memes. In this way we design a MSXF+B (MSXF with blocks) operator.

The neighborhood $\mathcal{N}(\pi)$ of the permutation (individual) π is defined as a set of new permutations that can be achieved from π by exactly one adjacent pairwise exchange operator which exchanges the positions of two adjacent jobs of a problem's solution connected with permutation π. The distance measure $d(\pi,\sigma)$ is defined as a number of adjacent pairwise exchanges needed to transform permutation π into permutation σ. Such a measure is known as Kendall's τ measure. The outline of the procedure is presented on the Fig. 2.

```
Algorithm 2.  Multi-Step Crossover
                Fusion with Blocks
Let π_1, π_2 be parent solutions. Set x = q = π_1;
repeat
  Determine blocks in the solution π.
  Determine restricted neighborhood N(π)
    according to blocks;
  For each member y_i ∈ N(π) calculate d(y_i, π_2);
  Sort y_i ∈ N(π) in ascending order of d(y_i, π_2);
  repeat
    Select y_i from N(π) with a probability
      inversely proportional to the index i;
      Calculate C_sum(y_i);
    Accept y_i with probability 1,
      if C_sum(y_i)  ≤ C_sum(x), and with
      probability P_T(y_i) = exp((C_sum(x)−
      −C_sum(y_i)) / T) otherwise
      (T is temperature);
    Change the index of y_i from i to n and the
      indices of y_k, k = i+1,...,n from k to k−1;
  until y_i is accepted;
  x ← y_i;
  if C_sum(x) < C_sum(q) then
    q ← x;
until some termination condition is satisfied;
q is the offspring.
```

Figure 2: Outline of the Multi-Step Crossover Fusion with Blocks procedure.

In the implementation proposed here Multi-Step Crossover Fusion with Blocks (MSXF+B) is an inter-island (i.e. inter–subpopulation) crossover operator which constructs a new individual by making use of the best individuals of different islands connected with subpopulations on different processors. The condition of termination consisted in exceeding 100 iterations by the MSXF+B function. The outline of the whole parallel memetic algorithm is presented on the Fig. 3.

```
Algorithm 3.  Parallel memetic algorithm
parfor j = 1,2,...,p   { p - #processors }
  i ← 0;
  P_j ← random subpopulation connected
    with processor j;
  p_j ← number of individuals in
    j-th subpopulation;
  repeat
    Selection(P_j, P'_j);
    Crossover(P'_j, P''_j);
    Mutation(P''_j);
    if (k mod R = 0) then {every R iteration}
      r := random(1, p);
      MSXF+B(P'_j(1), P_r(1));
    end if;
    P_j ← P''_j; i ← i + 1;
    if there is no improvement of the
      average C_sum then
      {Partial restart}
      r ← random(1,p);
      Remove α = 90 percentage of individuals
        in the subpopulation P_j.;
      Replenish P_j by random individuals;
    end if;
    if (k mod S = 0) then {Migration}
      r ← random(1,p);
      Remove β = 20 percentage of individuals
        in the subpopulation P_j;
      Replenish P_j by the best individuals
        from the subpopulation P_r
      taken from processor r;
    end if;
  until Stop_Condition;
end parfor
```

Figure 3: Outline of the parallel memetic algorithm.

4 COMPUTER SIMULATIONS

The algorithm was implemented in the Ada95 language and ran on the SGI Altix 3700 Bx2 supercomputer installed in Wrocław Center of Networking and Supercomputing under the Novell SUSE Linux Enterprise Server operating system. Tests were based on 125 instances with 40,50 and 100 jobs taken from the OR-Library (http://people.brunel.ac.uk/∼mastjjb/jeb/info.html). The results were compared to the best known, also taken from OR-Library.

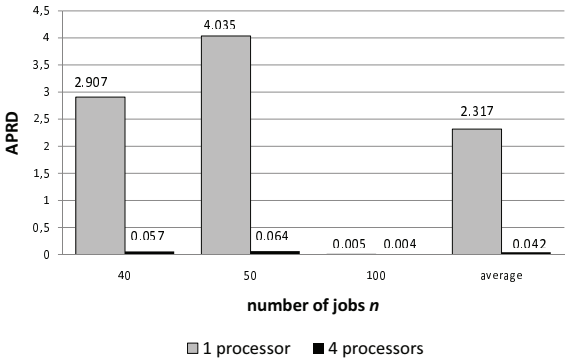

Figure 4: Average percentage relative deviations (APRD) to the best known solutions for the sequence and parallel memetic algorithms.

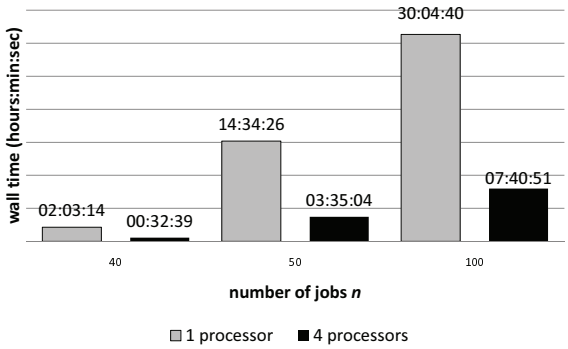

Figure 5: Computing times.

The computational results as well as computing times are presented on Figures 4 and 5. The number of iterations was counted as a sum of iterations on processors, and permanently set to 800. For example, 4-processor implementations make 200 iterations on each of the 4 processors, so we can obtain comparable costs of computations. As we can observe, the parallel versions of the algorithm achieve much better results of the average and maximal relative deviation from the optimal (or the best known) solutions, working (parallel) in a shorter time. Due to the small cost of communication the speedup parameter of the parallel algorithms is almost linear.

5 REMARKS AND CONCLUSIONS

The Lamarck evolution theory as well as memetic approach not only significantly extend traditional GA, but offers more effective approach, too. It is well

known, that the classic GA has a week search intensification phase – genetic operators as well as a mutation mainly diversify the search process. Additionally, in the memetic approach it is possible to make use of specific problem properties such as the new MSXF+B operator with block properties. Embedding special properties of the problem inside GA is usually difficult. Further benefits are obtained by using an island model with inter-island operator for the parallel asynchronous coevolution.

As we observe MA is also able to improve convergence time comparing to GA. Compared to a sequential algorithm, the parallelization of MA shortens the computations time and improves quality of obtained solutions. The proposed methodology of memetic algorithms parallelization can be applied to solve concurrently all scheduling problems with block properties, such as flow shop and job shop problems with makespan criterion, single machine scheduling problems, etc., for which a solution is represented as a permutation.

ACKNOWLEDGEMENTS

The work was partially supported by the Polish Ministry of Science and Higher Education, grant No. N N514 470439.

REFERENCES

Baker, K. and Scudder, G. (1990). Sequencing with earliness and tardiness penalties: a review. *Operations Research*, 38:22–36.

Berger, J. and Barkaoui, M. (2002). A memetic algorithm for the vehicle routing problem with time windows. In *Proceedings of the 7th International Command and Control Research and Technology Symposium*, http://www.dodccrp.org/events/7th_ICCRTS/Tracks/pdf/035.pdf.

Bożejko, W. (2010). Parallel path relinking method for the single machine total weighted tardiness problem with sequence-dependent setups. *Journal of Intelligent Manufacturing, in press, doi: 10.1007/s10845-009-0253-2*.

Bożejko, W., Grabowski, J., and Wodecki, M. (2006). Block approach tabu search algorithm for single machine total weighted tardiness problem. *Computers & Industrial Engineering*, 50:1–14.

Bożejko, W. and Wodecki, M. (2005). Task realiation's optimization with earliness and tardiness penalties in distributed computation systems. volume 2528 of *Lecture Notes in Artificial Intelligence*, pages 69–75. Springer.

Bożejko, W. and Wodecki, M. (2006). A new inter-island genetic operator for optimization problems with block properties. volume 4029 of *Lecture Notes in Artificial Intelligence*, pages 324–333. Springer.

Bradwell, R. and Brown, K. (1999). Parallel asynchronous memetic algorithms. In Cantu-Paz, E. and Punch, B., editors, *Evolutionary Computation*, pages 157–159.

Bubak, M. and Sowa, K. (1999). Objectoriented implementation of parallel genetic algorithms. In Buyya, R., editor, *High Performance Cluster Computing: Programming and Applications*, volume 2, pages 331–349. Prentice Hall.

Crainic, T. and Toulouse, M. (1998). Parallel metaheuristics. In Crainic, T. and Laporte, G., editors, *Fleet management and logistics*, pages 205–251. Kluwer.

Crainic, T., Toulouse, M., and Gendreau, M. (1995). Parallel asynchronous tabu search in multicommodity locationallocation with balancing requirements. *Annals of Operations Research*, 63:277–299.

Hoogeveen, J. and van de Velde, S. (1996). A branch and bound algorithm for single-machine earliness-tardiness scheduling with idle time. *INFORMS Journal on Computing*, 8:402–412.

Kliewer, G., Klohs, K., and Tschoke, S. (1999). Parallel simulated annealing library: User manual. Technical report, Computer Science Department, University of Paderborn.

Lenstra, J., Kan, A. R., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag.

Reeves, C. and Yamada, T. (1998). Genetic algorithms, path relinking and the flowshop sequencing problem. *Evolutionary Computation*, 6:45–60.

Smith, W. (1956). Various optimizers for single-stage production. *Naval Research Logistic Quart*, 3:59–66.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17:443–455.

Taillard, E. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6:108–117.

Tang, J., Lim, M., and Ong, Y. (2006). Adaptation for parallel memetic algorithm based on population entropy. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (Seattle, Washington, USA, July 08 - 12, 2006), GECCO '06*, pages 575–582. ACM.