Algorithmica (2015) 73:21–41
DOI 10.1007/s00453-014-9898-0

Performance Analysis of the (1+1) Evolutionary Algorithm for the Multiprocessor Scheduling Problem

Yuren Zhou · Jun Zhang · Yong Wang

Received: 11 August 2013 / Accepted: 21 May 2014 / Published online: 5 June 2014
© Springer Science+Business Media New York 2014

Abstract In recent years, there has been considerable progress in the theoretical study of evolutionary algorithms (EAs) for discrete optimization problems. However, results on the performance analysis of EAs for NP-hard problems are rare. This paper contributes a theoretical understanding of EAs on the NP-hard multiprocessor scheduling problem. The worst-case bound on the (1+1)EA for the multiprocessor scheduling problem and a worst-case example are presented. It is proved that the (1+1)EA on $Q2 \mid\mid C_{\max}$ problem achieves an approximation ratio of $\frac{1+\sqrt{5}}{2}$ in expected time $O(n^2)$. Finally, the theoretical analysis on three selected instances of the multiprocessor scheduling problem shows that EAs outperform local search algorithms on these instances.

Keywords Evolutionary algorithms · Scheduling problem · Local search algorithms · Approximation algorithms · Analysis of algorithms

Y. Zhou (✉)
School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China
e-mail: yrzhou@scut.edu.cn

J. Zhang
Department of Computer Science, Sun Yat-Sen University, Guangzhou 510275, China
e-mail: issai@mail.sysu.edu.cn

Y. Wang
College of Information Science and Engineering, Central South University, Changsha, Hunan, China
e-mail: ywang@csu.edu.cn

1 Introduction

Evolutionary algorithms are a class of randomized heuristics that are based on analogies to natural evolution [1]. They have been widely applied to various complex real-world problems. The theoretical study of evolutionary algorithms has received much interest in recent years [2–4]. There is a common belief that a solid foundation for such heuristics is needed, which would improve the understanding of the algorithms, guide the choice of the parameters, etc..

A major part of the theoretical research of evolutionary algorithms is the rigorous runtime analysis that stems from the theoretical computer science and the probabilistic analysis of randomized algorithms [5,6]. Oliveto et al. [3] gave a comprehensive survey of the time complexity analysis of evolutionary algorithms up to 2007. For more information and details, we refer the reader to the text book of Neumann and Witt [4].

Initially, the runtime analysis was carried out on the simplest evolutionary algorithm called (1+1)EA for different kinds of artificial pseudo-Boolean functions such as linear function, Leadingones, Trap functions etc. [7,8]. The study on these functions with some structural properties is helpful to understand the behavior of evolutionary algorithms and provides useful mathematical methods and tools for the analysis on realistic problems. Later classical combinatorial optimization problems in the P class such as maximum matchings [9], minimum spanning tree [10], shortest path [11,12] etc. were considered. The analysis shows that evolutionary algorithms can solve classical polynomial-time problems efficiently in the expected polynomial time, although they can not outperform the best problem-specific algorithms for these problems.

The population is the main element of evolutionary algorithms and there have been theoretical investigations on the impact of different population sizes on the performance of EAs [13–16]. The analyses show that a large population may be helpful or harmful for EAs on different instances.

In practice, many interesting discrete optimization problems are NP-hard [17]. Complexity theory tells us that there is no polynomial-time exact algorithm for such problems unless $P = NP$, and this is very unlikely to be true. A natural question arises whether approximate solutions can be found efficiently for such hard optimization problems. An algorithm for a minimization problem is called a α-approximation algorithm if the algorithm produces a solution that has value at most α times the optimal solution value.

Although evolutionary algorithms are claimed to be a class of global optimization algorithms, one can not hope that they solve every instance of the NP-hard problem in (expected) polynomial time. While evolutionary algorithms may be very efficient in practice for obtaining near optimal solutions for the NP-hard problem, it is natural to perform the approximation analysis which provides guarantees on the quality of solutions obtained in the worst case.

One of the first performance analyses of evolutionary algorithms for NP-hard problems is by Witt [18], who investigated the performance guarantees of the (1+1)EA and the randomized local search algorithm (RLS) on the partition problem, which is equivalent to the simple case of the scheduling problem with two identical machines($P2 \mid\mid C_{\max}$). He proved that both algorithms achieve an approximation

ratio of $\frac{4}{3}$ for the partition problem in expected runtime $O(n^2)$, and a variant of the (1+1)EA with restarts is a PRAS (polynomial-time randomized approximation scheme), i.e. it gives a $(1+\epsilon)$-approximation in expected time that depends polynomially on n and $\epsilon > 0$. An average-case analysis with respect to certain input distributions showed that the (1+1)EA finds solutions that converge to optimality in expectation. Gunia [19] later extended Witt's result to $k-$partition problem. He showed that the (1+1)EA and RLS create a $(2k/(k+1))-$approximation on this problem in an expected pseudo-polynomial runtime. Nevertheless, both algorithms take an expected exponential running time to find $(2k/(k+1)-\epsilon)-$approximation in the worst case for a constant $\epsilon > 0$. Recently, Sutton and Neumann [20] studied multi-start evolutionary algorithms for the makespan scheduling on two machines by carrying out parameterized runtime analyses. They showed that multi-start variants of the (1+1) EA and RLS are Monte Carlo fpt-algorithms for a parameterization which considers the value of the optimal solution above its lower bound.

Friedrich et al. [21] presented the theoretical analyses on the hybrid method combining evolutionary algorithms with approximation algorithms for the vertex cover problem. They gave several specific instances to show that approximation solutions can (or cannot) be improved by evolutionary algorithms. Recently, Yu et al. [22] studied an EA framework named simple EA with isolated population (SEIP) and showed it achieves a H_k-approximation ratio (where H_k is the harmonic number of the cardinality of the largest set) for the unbounded set cover problem, and a $H_k - \frac{k-1}{8k^9}$-approximation ratio for the k-set cover problem in expected polynomial times. Sutton and Neumann [23] investigated the randomized local search and the (1+1)EA using 2-opt mutation for the traveling salesman problem with few inner points. They showed that under certain geometric constrains, evolutionary algorithms can solve the convex TSP, and randomized local search algorithms find local optima both in polynomial times.

The scheduling problems have been the subject of extensive research for over 50 years [24,25]. Most scheduling problems are NP-hard, even in the simple case with two machines [17]. As a result, there have been many methods based on the heuristic search to obtain optimal and sub-optimal solutions to the problems. The multiprocessor scheduling problem is a basic and well studied NP-hard scheduling problem [26–31]. It has a wide range of applications which include assembly of printed circuit boards [32], design of flexible manufacturing systems [33] and efficient execution of computing tasks on server systems [34]. In this paper, we carry out a performance analysis for the evolutionary algorithm on the multiprocessor scheduling problem. We use the optional stopping theorem to prove that the (1+1)EA, as well as the well-known swap local search algorithm [27,35], can achieve an expected polynomial time approximation ratio of $\frac{1+\sqrt{5}}{2}$ for $Q2 \mid\mid C_{max}$ problem, and this ratio is almost tight. We present three selected instances of different types (namely, $P \mid\mid C_{\max}$, $Q \mid\mid C_{\max}$, and $R2 \mid\mid C_{\max}$) of the multiprocessor scheduling problem, in which local search algorithms may get trapped in local optima, and we show that evolutionary algorithms can solve them efficiently in expected polynomial time. We also present the worst-case bound on the (1+1)EA for the multiprocessor scheduling problem and a worst case example.

The rest of this paper is organized as follows. Section 2 introduces the problem, algorithms and tools for the analysis of algorithms. Section 3 presents the worst-case bounds of the (1+1)EA for the multiprocessor scheduling problem and proves the performance guarantee of the (1+1)EA on $Q2 \mid\mid C_{max}$ problem. Section 4 shows that the (1+1)EA outperforms local search algorithms on three selected instances. Section 5 concludes the paper with a discussion of the results.

2 The Problem, Algorithms and Tools

In this section, we briefly describe the formulation of the multiprocessor scheduling problem, local search and evolutionary algorithms for the multiprocessor scheduling problem, and tools for the analysis of EAs.

2.1 The Multiprocessor Scheduling Problem

We consider the classical multiprocessor scheduling problem. We are given a set of n jobs $J = \{J_1, \cdots, J_n\}$ and a set of m machines $M = \{M_1, \cdots, M_m\}$. Each job needs to be scheduled on a machine, and a machine can process at most one job at a time. The processing time of job J_j on machine M_i is denoted by p_{ij}. The load on a machine is the sum of the processing times of the jobs running on it. The machine with the maximum load is called a $critical$ machine. The objective of multiprocessor scheduling is to determine a schedule of jobs on the machines to minimize the makespan, i.e., to minimize the maximum loads of machines.

There are usually three types of machine environment: identical parallel machines, uniform parallel machines and unrelated parallel machines.

In the identical parallel machines environment, denoted by P, the time required for processing job J_j is equal on all machines, i.e. $p_{ij} = p_j, \forall i \in \{1, \ldots, m\}$. In the uniform parallel machines environment, denoted by Q, each job J_j has a processing requirement p_j and each machine M_i has speed s_i, and the time that it takes machine M_i to process job J_j is $p_{ij} = p_j/s_i$. In the unrelated parallel machines environment, denoted by R, the processing time p_{ij} may be arbitrary positive numbers.

The above classification scheme is introduced by Graham et al. [24]. When the number of machines is part of the input, they denote these problems by $P \mid\mid C_{\max}$, $Q \mid\mid C_{\max}$, and $R \mid\mid C_{\max}$ respectively. When the number of machines is a constant m, these problems are denoted by $Pm \mid\mid C_{\max}$, $Qm \mid\mid C_{\max}$, and $Rm \mid\mid C_{\max}$, respectively.

It is well known that all these problems with $m \geq 2$ are NP-hard [17]. For $P \mid\mid C_{\max}$ and $Q \mid\mid C_{\max}$ problems, Hochbaum and Shmoys [29] presented a polynomial dual algorithm that guarantees a relative error of ϵ in time $O((n/\epsilon)^{1/\epsilon^2})$. For $R \mid\mid C_{max}$ problem, Lenstra et al. [30] proposed a polynomial 2-approximation algorithm, and demonstrate that no polynomial algorithm can achieve an approximation ratio less than $3/2$ unless $P = NP$.

2.2 Local Search Algorithms for the Multiprocessor Problem

A local search algorithm is an iterative algorithm that transforms one solution to another according to some neighborhood structure. It starts from some given initial

Table 1 The approximation ratio α obtained by local search (from [31])

	Jump	Swap	Push
$P2 \mid\mid C_{\max}$	4/3 [26]	4/3 [26]	8/7 [31]
$P \mid\mid C_{\max}$	$2 - \frac{2}{m+1}$ [26]	$2 - \frac{2}{m+1}$ [26]	$\frac{4m}{3m+1} \le \alpha \le 2 - \frac{2}{m+1}$ [31]
$Q2 \mid\mid C_{\max}$	$\frac{1+\sqrt{5}}{2}$ [27]	$\frac{1+\sqrt{5}}{2}$ [27]	$\frac{1+\sqrt{17}}{4}$ [31]
$Q \mid\mid C_{\max}$	$\frac{1+\sqrt{4m-3}}{2}$ [27]	$\frac{1+\sqrt{4m-3}}{2}$ [27]	$\frac{3}{2} - \epsilon \le \alpha \le 2 - \frac{2}{m+1}$ [31]
$R2 \mid\mid C_{\max}$	$\frac{p_{\max}}{C^*_{\max}} \le \alpha$ [31]	$n - 1 \le \alpha$ [31]	Undefined
$R \mid\mid C_{\max}$	$\frac{p_{\max}}{C^*_{\max}} \le \alpha$ [31]	$\frac{p_{max}}{C^*_{\max}} \le \alpha$ [31]	Undefined

solution and then iteratively tries to find a better solution in an appropriately defined neighborhood of the current solution. The local search algorithm stops at a locally optimum solution when no better solution is found.

For the multiprocessor problem, there are typically three neighborhoods based on jump, swap and push moves. A jump (also called move) neighbor is generated by moving a job to a machine on which it is not scheduled. The swap neighbor is obtained by interchanging the machine allocations of any two jobs which are not scheduled on the same machine. The push neighborhood is a more complex one that is introduced in [31]. The approximation ratios for local search algorithms with these neighborhoods are summarized in Table 1.

2.3 The Evolutionary Algorithm for the Multiprocessor Scheduling Problem

An evolutionary algorithm is an iterative procedure which borrows the ideas of natural selection and genetic evolution. It uses the principles of selection, reproduction and survival of the fittest as a means for evaluating regions of the search space.

An evolutionary algorithm usually works with a population of potential solutions which are called chromosomes in analogy to natural genetics, and uses a fitness value to guide the search. For multiprocessor scheduling problem, a feasible solution is given by a partition of job set J into m disjoint sets. In order to encode a solution of the multiprocessor scheduling problem, we define variables

$$x_{ij} = \begin{cases} 1, & \text{if job } J_j \text{ is assigned to machine } M_i, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Hence, the multiprocessor scheduling problem is formulated as

$$\begin{aligned} &\text{Minimize } \max_i l_i, \\ &\text{subject to } \sum_{i=1}^{m} x_{ij} = 1,\ j = 1, \ldots, n, \end{aligned}$$

where $l_i = \sum_{j=1}^{n} x_{ij} p_{ij}$ is the load of machine M_i and $x_{ij} \in \{0, 1\}$.

We make use of the vector schema to present variables x_{ij} and the objective function.

Let $x = (x_{11}, \ldots, x_{1n}, x_{21}, \ldots, x_{2n}, \ldots, x_{m1}, \ldots, x_{mn})$ be the row vector whose elements are those of matrix $(x_{ij})_{m\times n}$ in row order, and let $makespan(x) = \max\limits_{i} \sum_{j=1}^{n} x_{ij} p_{ij}$ be the makespan for the schedule corresponding x. Our objective is to minimize $makespan(x)$ on search space

$$S^* = \{x \mid x \text{ is the row vector of matrix } (x_{ij})_{m\times n} \text{in row order,}$$
$$x_{ij} = 0 \text{ or } 1, \sum_{i=1}^{m} x_{ij} = 1\}.$$

We now describe the (1+1)EA, the perhaps simplest evolutionary algorithm using mutation and selection approaches with population size of 1. The (1+1)EA is a simple but effective random hill-climbing EA. Assume that $f : S = \{0, 1\}^N \to R$ is the objective or fitness function to be minimized. The (1+1)EA can be formalized as follows.

Algorithm 1 The (1+1) EA
begin
initialization: Choose randomly an initial bit string x from search space S ;
while (termination-condition does not hold) do
Mutation: x' = mutate(x);
Selection: Replace x by x' iff $f(x') < f(x)$.
od
end

Note that the (1+1)EA given in this paper only accepts strict improvements while many versions of the (1+1)EA accept new solutions of the same fitness [4].

For the multiprocessor scheduling problem, the (1+1)EA generates an initial solution randomly, i.e., each job is assigned uniformly at random to each machine. The algorithm creates new solution (offspring) x' from old solution x by mutation as follows. Each job is scheduled on the same machine with probability $1 - p_m$ (parameter p_m is called the mutation probability and its generic value equals $1/n$), and leaves its current machine with probability p_m. Once a job leaves its current machine, it moves to any other machine randomly. Hence, for job J_j of x on machine M_i, it is scheduled on the same machine M_i with probability $1 - p_m$, and it moves to machine $M_k (k \neq i)$ with probability $p_m/(m - 1)$.

2.4 The Martingale and Optional Stopping Theorem

Martingales are considered to be very useful tools for studying stochastic process and have numerous applications of stochastic modeling. Details on this topic can be found in any literature regarding random processes, e.g. [5,6]. The search processes introduced by evolutionary algorithms may be modeled as (super)martingales. In this paper we make use of martingales to prove our main result (Theorem 4).

Martingales are stochastic processes that represent the notion of a fair game in the context of gambling. In a fair game, the gambler's fortune on the next play is, on the average, his current fortune and not dependent upon the previous results.

The formal definition of a martingale is given below.

Definition 1 A sequence of random variables $Z_0, Z_1, \ldots$ is called a martingale with respect to the sequence of random variables $X_0, X_1, \ldots$ if, for all $n \geq 0$

(1) Z_n is a function of $X_0, X_1, \ldots, X_n$;
(2) $E(\mid Z_n \mid) < \infty$;
(3) $E(Z_{n+1} \mid X_0, \ldots, X_n) = Z_n$.

Definition 2 Let $X = \{X_n : n \geq 0\}$ be a sequence of random variables defined on the same probability space. A stopping time with respect to X is a random variable T taking values in $\{0, 1, \ldots\}$ such that for each $n \geq 0$, the event $\{T = n\}$ is completely determined by the information known up to the time n, $\{X_0, X_1, \ldots, X_n\}$.

For example, the first time the sequence $X_0, X_1, \ldots$ reaches a subset A of the state space is a stopping time.

The term "determined" in the definition means the indicator function of the event $\{T = n\}$ can be written as a function of $X_0, X_1, \cdots, X_n$, i.e. the occurrence or non-occurrence of the event $\{T = n\}$ depends only on the values of $X_0, X_1, \ldots, X_n$. An alternative and more general definition may be given in terms of the $\sigma-$fields [5].

The intuition behind the definition of the stopping time is that at any particular time n, you can look at the sequence so far and tell if it is time to stop.

The following theorem called optional stopping theorem is one of the central facts in the theory of martingale sequences [6].

Theorem 1 *(Optional stopping theorem) Suppose that $Z_0, Z_1, \ldots$ is a martingale with respect to the sequence of random variables $X_0, X_1, \ldots$ and T is a stopping time for $X_0, X_1, \ldots$. If one of the following conditions holds*

(1) Z_i is bounded, i.e. $\mid Z_i \mid \leq c$ for any $i \geq 0$ and some constant c,
(2) T is bounded,
(3) $E(T) < \infty$, and there is a constant c such that $E(\mid Z_{i+1} - Z_i \mid \mid X_0, X_1, \ldots, X_i) \leq c$ for all i, then $E(Z_T) = E(Z_0)$.

The optional stopping theorem shows that, under certain conditions, the expected value of a martingale at a stopping time is equal to the expectation of its initial random variable.

3 The Worst Case Bound on the (1+1)EA and the Approximation Guarantee of the (1+1)EA for $Q2 \mid\mid C_{\max}$ Problem

3.1 The Worst Case Bound and a Worst Case Example for the (1+1)EA on $Rm \mid\mid C_{\max}$ Problem

We consider the worst case runtime bound on the (1+1)EA for the multiprocessor scheduling problem. We present a general worst case upper bound for the expected

runtime of the (1+1)EA on any multiprocessor schedule instance, and describe a worst case $R \mid\mid C_{\max}$ example to show this upper bound is attainable.

Theorem 2 *The expected runtime of the (1+1)EA for any $R \mid\mid C_{max}$ instance with m machines and n jobs is at most $(n(m-1))^n$.*

Proof Let x and x^* represent the current solution and the globally optimal solution respectively. Since in one step of the (1+1)EA, a job remains on the same machine with probability $1-\frac{1}{n}$ and moves to any other machine with probability $\frac{1}{n(m-1)}$, the probability mutating from x to x^* in one step is at least $(\frac{1}{n(m-1)})^n$ which proves the theorem. □

For an $R \mid\mid C_{\max}$ instance, the processing time of job J_j on machine M_i is p_{ij}, and the maximum processing time is denoted as $p_{\max} = \max_{ij} p_{ij}$. The following example is given by Schuurman and Vredeveld [31]. By this instance, they derive a general lower bound on the performance ratio of the jump local search algorithm for $R \mid\mid C_{\max}$ problem.

$Example I_1$ For given $K > 1$, consider n jobs and $m = n$ machines. The processing times of jobs are

$$p_{ij} = \begin{cases} 1, & if\ \ i = j, \\ K, & otherwise. \end{cases}$$

In the optimal schedule with makespan one, job $J_i (i = 1, \ldots, n)$ is assigned to machine M_i. We now consider schedule σ_1 in which job $J_i (i = 1, \ldots, n-1)$ is assigned to machine M_{i+1} and J_n is assigned to M_1. This schedule is jump optimal and has makespan K.

By this instance, the general performance guarantee of the jump local search algorithm for $R \mid\mid C_{\max}$ cannot be smaller than $p_{\max}/C^*_{\max}$ [31], where $C^*_{\max}$ is the optimal makespan.

Searching the optimal solution of I_1 in the search space is similar to finding a needle in a big haystack. Such problem is difficult for any search algorithm.

Theorem 3 *Starting with schedule σ_1, the (1+1)EA on $R \mid\mid C_{max}$ instance I_1 takes on average $\Omega((n(n-1))^n)$ steps to reach the optimal schedule.*

Proof Note that the optimal schedule has to be reached directly by mutating σ_1 in one step. The probability for this event is $(\frac{1}{n(n-1)})^n$ which proves the theorem. □

3.2 The Performance Guarantee of the (1+1)EA for $Q2 \mid\mid C_{\max}$ Problem

We now study the approximation performance of the (1+1)EA on $Q2 \mid\mid C_{\max}$ problem. We show that the (1+1)EA reaches a solution with performance ratio at least $\frac{1+\sqrt{5}}{2}$ in expected time $O(n^2)$ and this bound is almost tight.

For given $Q2 \mid\mid C_{\max}$ instance Q, we have two machines M_1 and M_2, and n jobs $J_1, \ldots, J_n$ with processing requirements $p_1, \ldots, p_n$. We also define $p = \sum_{i=1}^{n} p_i$.

In the following, we assume that machine M_1 has a speed of 1 and M_2 has a speed of $s \geq 1$. Note that these assumptions do not affect the generality of the proofs. In order to simplify the notation, we define the search space $S = \{0, 1\}^n$. For a solution $x = (x_1, x_2, \ldots, x_n) \in S$, each bit x_i of x corresponds to job J_i. Job J_i is assigned to machine M_1 if $x_1 = 1$ and otherwise it is assigned to machine M_2. This representation differs from the one introduced in Sect. 2.3. Now every bit string corresponds to a feasible schedule.

Given the above representation, the loads of M_1 and M_2 for a search point $x = (x_1, x_2, \cdots, x_n) \in S$ are $l_1(x) = \sum_{i=1}^{n} x_i p_i$ and $l_2(x) = p - \sum_{i=1}^{n} x_i p_i$, respectively. The fitness of x is given by

$$fit(x) = max\{t_1(x), t_2(x)\}, \tag{2}$$

where $t_1(x) = l_1(x)$ and $t_2(x) = l_2(x)/s$ are the processing times of M_1 and M_2 respectively.

Let $\rho_0 = \frac{1+\sqrt{5}}{2}$. Note that the definition of ρ_0 implies $1 + \frac{1}{\rho_0} = \rho_0$.

The following theorem, which is proven using the optional stopping theorem, bounds the approximation ratio of the (1+1)EA on the $Q2 \mid\mid C_{\max}$ problem.

Theorem 4 *For any $Q2 \mid\mid C_{max}$ instance Q, let $t^*(Q)$ be the optimal makespan of Q. Assume that speed s is a constant (independent of n), then the (1+1)EA on Q reaches a solution with makespan at most $\rho_0 t^*(Q)$ in an expected number of $O(n^2)$ steps.*

The objective of the (1+1)EA is to find a solution that has the optimal(smallest) fitness value. We denote by $X_t (t = 0, 1, \ldots)$ the random string describing at which the (1+1)EA on Q is during iteration t. Suppose at iteration t, machine M_1 defines the fitness value, i.e. $fit(X_t) = t_1(X_t)$. Let $g(X_t) = t_1(X_t) - t_2(X_t)$ be the difference of the finish times for the slower machine M_1 minus the faster machine M_2. If there exists a job on machine M_1 whose processing time is less than $g(X_t)$, then moving this job from M_1 to M_2 will decrease the fitness function value. Intuitively, if this reduction is always positive, we may expect that the fitness function reaches a value no greater than $\rho_0 t^*(Q)$ at some time.

We partition the search space S into two sets, the approximation set $S_1 = \{x \in S \mid fit(x) \leq \rho_0 t^*(Q)\}$ and its complement set $S_2 = \{x \in S \mid fit(x) > \rho_0 t^*(Q)\}$, and define the indicator function of S_2 as

$$1_{S_2}(x) = \begin{cases} 1, & if\ x \in S_2, \\ 0, & if\ x \in S_1. \end{cases}$$

For sake of the proof of Theorem 4, we now define the distance between $x \in S$ and the approximation set S_1 as

$$d(x, S_1) = \max\{fit(x) - \rho_0 t^*(Q), 0\} + p 1_{S_2}(x), \tag{3}$$

where $p = \sum_{i=1}^{n} p_i$ is the sum of all processing requirements. To be concise, we denote the distance by $d(x)$. The first term of the sum in (3) corresponds to the fitness

function value, and a penalty-like term is added to penalize the solution in S_2. Clearly, $d(x) = 0$ for any $x \in S_1$ and $d(x) > p$ for any $x \in S_2$.

In the following, we show that the random sequence $d(X_0), d(X_1), \ldots$ produced by the (1+1)EA on Q reaches zero, the smallest distance value in expected time $O(n^2)$, which will prove the theorem.

Proof (Theorem 4)

For $t = 0, 1, \ldots$, let $Y_t = d(X_t)$. Then $Z_t = \sum_{i=1}^{t}(Y_i - E(Y_i \mid Y_0, \ldots, Y_{i-1}))$ defines a martingale satisfying $E(Z_0) = 0$ (By convention, $E(Y_i \mid Y_0, \ldots, Y_{i-1}) = E(Y_i)$ when $i = 0$).

Indeed, for $t = 1, \ldots,$

$$\begin{aligned} &E(Z_{t+1} \mid Z_1, \ldots, Z_t) = E(Z_{t+1} \mid Y_0, \ldots, Y_t) \\ &= E(Z_t \mid Y_0, \ldots, Y_t) + E(Y_{t+1} - E(Y_{t+1} \mid Y_0, \ldots, Y_t) \mid Y_0, \ldots, Y_t) \\ &= Z_t + 0 = Z_t. \end{aligned}$$

We assume that the initial fitness value of the (1+1)EA is greater than $\rho_0 t^*(Q)$. Let T be the first time that the fitness value of the (1+1)EA is not greater than $\rho_0 t^*(Q)$, i.e. $T = min\{t : X_t \in S_1\}$. Note that random variable T is a stopping time.

We can rewrite Z_T as

$$\begin{aligned} Z_T &= Y_T - E(Y_T \mid Y_0, \ldots, Y_{T-1}) + \cdots + Y_1 - E(Y_1 \mid Y_0) + Y_0 - E(Y_0) \\ &= Y_T + \sum_{t=1}^{T}(Y_{t-1} - E(Y_t \mid Y_0, \ldots, Y_{t-1})) - E(Y_0) \end{aligned} \tag{4}$$

For $1 \le t \le T$, we claim that

$$Y_{t-1} - E(Y_t \mid Y_0, \ldots, Y_{t-1}) \ge \Omega\left(\frac{p}{n^2}\right). \tag{5}$$

We define $t_s = \frac{p}{1+s}$. Note that t_s is the theoretically smallest finish time attainable, i.e.

$$t_s \le t^*(Q). \tag{6}$$

We prove the above claim by considering two cases.

Case 1 $s \ge \rho_0$.

We argue that even when all jobs are assigned to machine M_2, $t_2(X_t)$ the processing time of M_2 at any time t satisfies

$$t_2(X_t) \le \rho_0 t^*(Q). \tag{7}$$

Indeed,

$$\begin{aligned} t_2(X_t) \le \frac{p}{s} &= \frac{1+s}{s}\frac{p}{1+s} \\ &\le \left(1+\frac{1}{s}\right)t^*(Q) \qquad \text{(by(6))} \\ &\le \left(1+\frac{1}{\rho_0}\right)t^*(Q) = \rho_0 t^*(Q). \qquad \text{(by assumption } \ s \ge \rho_0) \end{aligned}$$

For $1 \le t < T$, we have $fit(X_t) > \rho_0 t^*(Q)$, which implies the critical machine at time t is machine M_1. Hence, moving any job scheduled on M_1 to M_2 will decrease the fitness function value at time t.

At iteration t, we denote by A_1 the event that there exists a job J_i on M_1 with processing time p_i satisfying $p_i \ge \frac{s}{1+s}(t_1(X_t) - t_2(X_t))$ and A_2 the complement event of A_1. We also use $1_{A_i}(i = 1, 2)$ to represent the indicator random variable of event A_i.

If A_1 happens, we have $t_1(X_t) - p_i \le t_2(X_t) + \frac{p_i}{s}$. Thereafter, moving J_i from M_1 to M_2 implies $X_{t+1} \in S_1$ [by (7)], and the distance value decreases at least p. Since the probability of the event that a job moves from M_1 to M_2 while keeping the other $n-1$ jobs unchanged is $\frac{1}{n}(1-\frac{1}{n})^{n-1} \ge \frac{1}{en}$, we get

$$d(X_t) - E(d(X_{t+1}) \mid d(X_t)1_{A_1}) \ge \frac{p}{en}. \tag{8}$$

If A_2 happens, the processing time of any job $J_j \in M_1$ satisfies $p_j < \frac{s}{1+s}(t_1(X_t) - t_2(X_t))$, i.e. $t_1(X_t) - p_j > t_2(X_t) + \frac{p_j}{s}$. Thereafter moving any job J_j from M_1 to M_2 implies that the distance value is decreased at least p_j. Hence, we have

$$d(X_t) - E(d(X_{t+1}) \mid d(X_t)1_{A_2}) \ge \frac{1}{en}\frac{1}{\mid M_1 \mid}\sum_{J_j \in M_1} p_j,$$

where $\mid M_1 \mid$ denotes the cardinality of M_1 and $\mid M_1 \mid \le n$.

Noting at iteration t,

$$\begin{aligned} \sum_{J_j \in M_1} p_j &= t_1(X_t) > \rho_0 t^*(Q) \\ &\ge \rho_0 \frac{1}{1+s} p, \qquad \text{(by (6))} \end{aligned}$$

we get

$$d(X_t) - E(d(X_{t+1}) \mid d(X_t)1_{A_2}) \ge \frac{\rho_0}{e(1+s)}\frac{p}{n^2}. \tag{9}$$

Case 2 $s < \rho_0$.

The situation of Case 2 is much more complicated.

For $1 \le t < T$, we consider two subcases.

Case 2.1 Machine M_2 defines the fitness function at iteration t, i.e. $fit(X_t) = t_2(X_t) > \rho_0 t^*(Q)$.

We claim that any job J_i on M_2 with p_i satisfies $p_i \le (\rho_0 - 1)p$. Otherwise, there exists a job J_j on M_2 with $p_j > (\rho_0 - 1)p$. We have $t^*(Q) > \frac{(\rho_0-1)p}{s}$. Therefore we obtain

$$t_2(X_t) \le \frac{p}{s} = \frac{p}{s}\frac{\rho_0}{\rho_0} = \frac{p\rho_0}{s}(\rho_0 - 1) < \rho_0 t^*(Q), \tag{10}$$

which is a contradiction.

Since, by assumption, $t_2(X_t) > \rho_0 t^*(Q)$, the load of M_2 at iteration t is at least $s\rho_0 t^*(Q) \ge \frac{s}{1+s}\rho_0 p$(by (6)), and the load of M_1 is at most $p - \frac{s}{1+s}\rho_0 p$.

For any job $J_i \in M_2$ with $p_i \le (\rho_0 - 1)p$, using the above result we have

$$t_1(X_t) + p_i \le \left(p - \frac{s}{1+s}\rho_0 p\right) + (\rho_0 - 1)p = \rho_0 \frac{p}{1+s} \le \rho_0 t^*(Q). \tag{11}$$

Similar to Case 1, at iteration t, let B_1 be the event that there exists a job J_j on M_2 with $p_j > \frac{s}{1+s}(t_2(X_t) - t_1(X_t))$ and B_2 the complement event of B_1. And denote by $1_{B_i}(i = 1, 2)$ the indicator random variable of event B_i.

If B_1 happens, there exists a job J_j on M_2 with p_j satisfying $t_2(X_t) - \frac{p_j}{s} < t_1(X_t) + p_j$. Using (11), after job J_j migrates from M_2 to M_1 then the processing times of M_1 and M_2 are $t_1(X_t) + p_j$ and $t_2(X_t) - \frac{p_j}{s}$ which satisfy $t_2(X_t) - \frac{p_j}{s} < t_1(X_t) + p_j \le \rho_0 t^*(Q)$. Therefore the reduction in the distance is at least p. Hence we get

$$d(X_t) - E(d(X_{t+1}) \mid d(X_t)1_{B_1}) \ge \frac{p}{en}. \tag{12}$$

If B_2 happens, for any job J_i on M_2, the processing time p_i satisfies $t_2(X_t) - \frac{p_j}{s} \ge t_1(X_t) + p_j$. Therefore, after a job J_i migrates from M_2 to M_1 the reduction in the distance is at least $\frac{p_i}{s}$. Similar to that of Case 1, we have

$$d(X_t) - E(d(X_{t+1}) \mid d(X_t)1_{B_2}) \ge \frac{\rho_0}{e \cdot s \cdot (1+\rho_0)}\frac{p}{n^2}. \tag{13}$$

Case 2.2 Machine M_1 defines the fitness function at iteration t, i.e. $fit(X_t) = t_1(X_t) > \rho_0 t^*(Q)$.

Similar to that of Case 1, we consider two possible events.

We denote by C_1 the event that there exists a job J_j on M_1 with processing time p_j satisfying $p_j \ge \frac{s}{1+s}(t_1(X_t) - t_2(X_t))$ and C_2 the complement event of C_1. We also use $1_{C_i}(i = 1, 2)$ to represent the indicator random variable of event C_i.

If C_1 happens, we further distinguish two subcases.

Case 2.2.1 $p_j \ge (\rho_0 - 1)p$.

It is easy to see $t^*(Q) \ge \frac{(\rho_0-1)p}{s}$.

If p_j moves from M_1 to M_2, then the processing time of M_2 is $t_2(X_t) + \frac{p_j}{s} \le \frac{p}{s} \le \rho_0 t^*(Q)$ [similar to (10)] and the processing time of M_1 is $t_1(X_t) - p_j \le p - p_j \le (2-\rho_0)p < \frac{\rho_0 p}{1+\rho_0} \le \frac{\rho_0 p}{1+s} \le \rho_0 t^*(Q)$. Therefore moving p_j from M_1 to M_2 decreases the distance value by at least p.

Case 2.2.2 $\frac{s}{1+s}(t_1(X_t) - t_2(X_t)) \leq p_j < (\rho_0 - 1)p$.
It is clear that

$$t_1(X_t) - p_j \leq t_2(X_t) + \frac{p_j}{s}. \tag{14}$$

By assumption, $t_1(X_t) = fit(X_t) > \rho_0 t^*(Q) \geq \frac{\rho_0 p}{1+s}$. Then the load of M_2 at iteration t is at most $p - \frac{\rho_0 p}{1+s}$ and $t_2(X_t) \leq \frac{1}{s}(p - \frac{\rho_0 p}{1+s})$.
If p_j migrates from M_1 to M_2, the processing time of M_2 is

$$t_2(X_t) + \frac{p_j}{s} \leq \frac{1}{s}\left(p - \frac{\rho_0 p}{1+s}\right) + \frac{(\rho_0 - 1)p}{s} = \frac{\rho_0 p}{1+s} \leq \rho_0 t^*(Q),$$

and the processing time of M_1 is $t_1(X_t) - p_j \leq t_2(X_t) + \frac{p_j}{s} \leq \rho_0 t^*(Q)$ [by (14)].
It follows that moving p_j from M_1 to M_2 reduces the distance value at least p.
Combining Cases 2.2.1 and 2.2.2, we obtain

$$d(X_t) - E(d(X_{t+1}) \mid d(X_t)1_{C_1}) \geq \frac{p}{en}. \tag{15}$$

If C_2 happens, the scenario is similar that of event A_2, and we have

$$d(X_t) - E(d(X_{t+1}) \mid d(X_t)1_{C_2}) \geq \frac{\rho_0}{e(1+\rho_0)}\frac{p}{n^2}. \tag{16}$$

Combining (8),(9),(12),(13),(15) and (16), we obtain (5).
Now, by (4) and (5), we have

$$Z_T \geq Y_T + \frac{cp}{n^2}T - E(Y_0),$$

where c is an appropriate constant.
Taking the expectation and applying optional stopping theorem yield

$$E(Y_T) + \frac{cp}{n^2}E(T) - E(Y_0) \leq E(Z_T) = E(Z_0).$$

Since $E(Y_T) = 0$, $E(Z_0) = 0$ and $E(Y_0) \leq p$, we obtain

$$E(T) = O(n^2).$$

This completes the proof. □

The key of the above proof lies in the estimate of the expected one-step distance decrease that is similar to one-step drift in [8]. However, in contrast to the drift analysis, we make use of optional stopping theorem to obtain the runtime bound. The above proof also implies that the result of drift analysis [8] may be deduced from optional stopping theorem.

Table 2 The approximation ratio obtained by the (1+1)EA

$P2 \mid\mid C_{\max}$	$Q2 \mid\mid C_{\max}$
4/3 [18]	$\frac{1+\sqrt{5}}{2}$ [Th. 4]

The approximation ratio ρ_0 in Theorem 4 that the (1+1)EA is able to obtain within expected polynomial time is almost tight, as can be seen in the following worst-case example.

$Example I_2$ Consider the $Q2 \mid\mid C_{\max}$ instance in which machine M_1 has speed one and machine M_2 has a speed of ρ_0. We have n jobs $J_1, \ldots, J_n$ with processing times $p_1 = (1-\epsilon)\rho_0$ and $p_i = \frac{1+\epsilon\rho_0}{n-1}$ for $i = 2, \ldots, n$, where ϵ is a constant satisfying $0 < \epsilon < \frac{\rho_0^2-1}{\rho_0^2+\rho_0}$. In the following, for simplicity, we assume that n is sufficiently large and $\frac{\epsilon\rho_0}{1+\epsilon\rho_0}(n-1)$ is an integer.

The optimal schedule for I_2 is obtained by assigning the large job J_1 and $\frac{\epsilon\rho_0}{1+\epsilon\rho_0}(n-1)$ small jobs to machine M_2 and by assigning all other $\frac{1}{1+\epsilon\rho_0}(n-1)$ jobs to M_1. The optimal makespan of I_2 is one.

Now consider schedule σ_2 in which the large job J_1 is assigned to M_1 and all other $n-1$ small jobs are assigned to M_2. The makespan for this schedule is $max\{(1-\epsilon)\rho_0, \frac{1+\epsilon\rho_0}{\rho_0}\} = (1-\epsilon)\rho_0$.

To decrease the makespan, the (1+1)EA has to move the large job J_1 from M_1 to M_2 and move at least $\frac{\epsilon\rho_0}{1+\epsilon\rho_0}(n-1)$ small jobs from M_2 to M_1. The probability that this event happens is bounded above by $n^{-\Omega(n)}$. Hence, starting from the aforementioned schedule σ_2, the (1+1)EA needs on average $n^{\Omega(n)}$ steps to decrease the makespan. We have now proven:

Theorem 5 *Starting with aforementioned schedule σ_2, the (1+1)EA on instance I_2 takes on average $n^{\Omega(n)}$ steps to create a solution better than $(1-\epsilon)\rho_0$ approximation.*

Table 2 summarizes the approximation ratio obtained by the (1+1)EA for multiprocessor scheduling problem.

4 The Analysis of the (1+1)EA on Three Selected Instances of the Multiprocessor Scheduling Problem

A local search algorithm starts with an initial solution and then searches the solution space by iteratively moving from one solution to a neighbor solution. Local search algorithms often get trapped in local optima. The evolutionary algorithm is usually considered a global optimization method that can escape local optima.

In this section, we describe three instances for the multiprocessor scheduling problem: $P \mid\mid C_{\max}$ instance I_3, $Q \mid\mid C_{\max}$ instance I_4 and $R2 \mid\mid C_{\max}$ instance I_5. We demonstrate that evolutionary algorithms can solve them in polynomial expected time, but the local search algorithms may get stuck in local optima. The rigorous theoretical

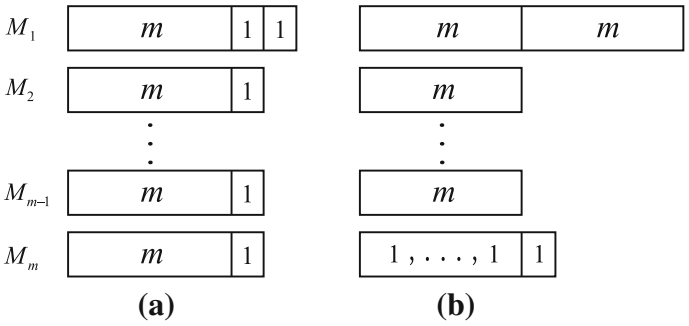

Fig. 1 **a** The optimal schedule for instance I_3. **b** The locally optimal schedule for the jump and swap local search algorithms on instance I_3

analysis shows that evolutionary algorithms can beat local searches on instances of the natural optimization problem.

Recall that in Sect. 2.3 a feasible solution (schedule) is presented by a row vector

$$x = (x_{11}, \ldots, x_{1n}, x_{21}, \ldots, x_{2n}, \ldots, x_{m1}, \ldots, x_{mn})$$

satisfying $\sum_{i=1}^{m} x_{ij} = 1$ and the makespan for x is $makespan(x) = \max\limits_{i} \sum_{j=1}^{n} x_{ij} p_j$.

The objective for the scheduling problem is to find a schedule to minimize the makespan. Recall that in Theorem 5, the (1+1)EA starting with schedule σ_1 needs on average exponentially many steps to solve $R \mid\mid C_{\max}$ instance I_2. This is because there are n machines defining the the makespan for σ_1 and the (1+1)EA needs to mutate simultaneously these machines to decrease the fitness value. Considering the case where more than one machine defines the makespan, we transform the objective function into a fitness function as

$$fit_1(x) = makespan(x) + \mid M(x) \mid, \tag{17}$$

where $M(x)$ is the set of machines defining the makespan of x.

In fitness function (17), a penalty-like term $M(x)$ is added to penalize the solution whose makespan is defined by more than one machine. By using fitness function (17), we will show that the runtime of EAs on three instances could be dramatically improved in Theorem 6 through 8 below.

We first present $P \mid\mid C_{\max}$ instance I_3 which is similar to the ones given by Michiels et al. [35] and Langston [28].

Example I_3 Consider m machines $M_1, \ldots, M_m$ and $n = 2m + 1$ jobs $J_1, \ldots, J_n$. The processing times of n jobs are $p_i = m$ for $i = 1, \ldots, m$ and $p_i = 1$ for $i = m + 1, \ldots, 2m + 1$. The job with processing time m (1) is called a large (small) job (respectively). The optimal makespan is $m + 2$. An optimal schedule is depicted in Fig. 1a in which machine M_1 processes one large job and two small jobs, and each of the other machines processes one large job and one small job.

In the schedule depicted in Fig. 1b, machine M_1 processes two large jobs, M_i processes one large job for $i = 2, \ldots, m-1$, and M_m processes $m+1$ small jobs. We can see that moving just one job or swapping any pair of jobs in this schedule does not decrease the makespan. This schedule is jump and swap optimal with makespan $2m$.

Contrary to the jump and the swap local search algorithms that may be trapped in a local optimum, the theorem below shows that the (1+1)EA on instance I_3 can reach the globally optimal solution within an expected polynomial time.

Theorem 6 *Starting with any initial solution, the expected running time of the (1+1)EA on instance I_3 is $O(n^4)$.*

Proof We partition search space $\{0,1\}^{mn}$ into four disjoint sets $A_i (i = 1, 2, 3, 4)$ with respect to different makespan values as follows

$$A_1 = \{x \in \{0,1\}^{mn} \mid x \text{ is a feasible solution, } makespan(x) > 2m\},$$
$$A_2 = \{x \in \{0,1\}^{mn} \mid x \text{ is a feasible solution, } makespan(x) = 2m\},$$
$$A_3 = \{x \in \{0,1\}^{mn} \mid x \text{ is a feasible solution, } m+2 < makespan(x) < 2m\},$$
$$A_4 = \{x \in \{0,1\}^{mn} \mid x \text{ is a feasible solution, } makespan(x) = m+2\}.$$

Note that during the run of the (1+1)EA the fitness value $fit_1(X_t)$ of the current string (solution) X_t never increases.

If the current string X_t belongs to A_1, we denote by $t_{min}(X_t) = \max\limits_i l_i$ the smallest load for the current schedule, where $l_i (i = 1, \ldots, m)$ is the load of machine M_i for X_t.

Recall that $makespan(X_t)$ is the largest load for X_t. Due to the definition of A_1, we have $makespan(X_t) \geq 2m+1$ and $t_{min}(X_t) \leq m$. Then, moving a job from a critical machine that defines $makespan(X_t)$ to a machine that defines $t_{min}(X_t)$ will decrease the fitness value by at least one. The probability for this event equals $\frac{1}{n(m-1)}(1-\frac{1}{n})^{n-1} \geq e^{-1}\frac{1}{n(m-1)}$, since a job moves from a machine to another specific machine with probability $\frac{1}{n(m-1)}$ and remains on the same machine with probability $1-\frac{1}{n}$.

Noting that the fitness value is a positive integer and not greater than m^2+m+1, we conclude that after expected time $O(nm^3)$ the (1+1)EA reaches $A_2 \cup A_3 \cup A_4$.

Now we assume the current string X_t belongs to A_2, i.e. $makespan(X_t) = 2m$. We distinguish two cases.

Case 1 There exists a critical machine that processes a large job and m small jobs.

Because $X_t \in A_2$, the shortest processing time $t_{min}(X_t)$ is at most m. Hence, moving a small job from the aforementioned critical machine to a machine that defines $t_{min}(X_t)$ decreases the fitness value by at least one, and the probability for this event is at least $e^{-1}\frac{1}{n(m-1)}$.

Case 2 Every critical machine processes two large jobs.

By pigeon hole principle, there exists a machine M_k that does not process any large job. Note that M_k processes at most $m+1$ small jobs.

If the load of machine M_k is less than m, then moving a large job from a critical machine to machine M_k decreases the fitness value by at least one, and again the probability for this event is at least $e^{-1}\frac{1}{n(m-1)}$.

If otherwise the load of M_k is not less than m (and at most $m+1$), then moving a large job from a critical machine to M_k and two small jobs from M_k to this critical machine will decrease the fitness value by at least one, and the probability for this event is bounded below by $\binom{m}{2}(\frac{1}{n(m-1)})^3(1-\frac{1}{n})^{n-3} \geq \frac{1}{2e}\frac{1}{n^3(m-1)}$.

Combining cases 1 and 2, we conclude again that after expected time $O(n^3m)$ the (1+1)EA reaches $A_3 \cup A_4$.

Once the (1+1)EA reaches A_3, due to the definition of set A_3, the critical machine for the current schedule processes a large job and l small jobs, where l is in the interval $[3, m-1]$. And the smallest load for the current schedule is at most $m+1$.

It follows that moving a small job from the aforementioned critical machine to a machine defining the smallest load will decrease the fitness value by at least one. Again the probability for this event is at least $e^{-1}\frac{1}{n(m-1)}$. Since the number of different fitness values in this phase is not greater than $2m$, the (1+1)EA reaches A_4 the global optimal within expected time $O(nm^2)$.

Altogether, we have proven the proposed upper bound. □

Cho and Sahni [27] showed that the list scheduling algorithm has a performance of $\frac{1+\sqrt{4m-3}}{2}$ for $Q \mid\mid C_{\max}$. Using the same techniques, Michiels et al. [35] proved the same performance guarantee for jump local search algorithm. The following example given by Schuurman and Vredeveld [31] shows that the aforementioned bound is tight.

Example I_4 Consider a $Q \mid\mid C_{\max}$ instance that has m machines and m jobs. M_1 has a speed of $s = \frac{1+\sqrt{4m-3}}{2}$ and all other machines have speed one. Note that m satisfies $m = s^2 - s + 1$. Job J_1 has a processing requirement of s and all other jobs have processing requirement one.

In the optimal schedule with makespan one, machine M_1 processes large job J_1 and each of the other machines processes one small job. Now consider schedule σ_3 in which machine M_2 processes large job J_1 and machine M_1 processes all other small jobs. Because moving any job in σ_3 does not decrease the makespan, σ_3 is locally optimal for the jump local search algorithm. The makespan for σ_3 is s, and the jump local search algorithm has performance ratio $s = \frac{1+\sqrt{4m-3}}{2}$.

Theorem 7 *Starting with any initial solution, the (1+1)EA using fitness function (17) for instance I_4 reaches the optimal schedule in expected runtime $O(m^4)$.*

Proof We divide the run of the (1+1)EA into three phases with respect to the fitness function (17): $fit_1(x) > s$, $fit_1(x) = s$ and $1 < fit_1(x) < s$, where x is the feasible solution (schedule).

During any time in the first phase, since the fitness is greater than s, the critical machine must be one of machines $M_2, \ldots, M_m$ and it processes at least two jobs. Then moving a job from the critical machine to an empty machine decreases the fitness value by at least one, and the probability for this event is at least $e^{-1}\frac{1}{m(m-1)}$. Noting the fitness function value is at most $s + (m-1)$, the (1+1)EA finishes the first phase in expected time $O(m^3)$.

During the time in the second phase, we consider two cases: there exists a critical machine that does not process job J_1 and otherwise. For the first case, the critical machine processes at least two small jobs, and moving a small job from the critical machine to an empty machine will decrease the fitness function value. For the second case in which the critical machine processes job J_1, we pessimistically assume that the current schedule is σ_3, then moving J_1 from the critical machine to M_1 and a small job from M_1 to any empty machine will decrease the fitness function value. The probability for this event is at least $e^{-1}\frac{1}{(m(m-1))^2}$. Therefore, the (1+1)EA finishes the second phase in expected time $O(m^4)$.

During any time in the third phase, job J_1 must be assigned to machine M_1, and any critical machine processes at least two jobs. Again, moving a small job from the critical machine to an empty machine decreases the fitness function value by at least $\frac{1}{s}$. The probability for this event is at least $e^{-1}\frac{1}{m(m-1)}$. Hence the (1+1)EA finishes the third phase in expected time $O(m^3)$.

Altogether the theorem follows. □

For unrelated parallel machines environment, Schuurman and Vredeveld [31] presented an instance of two machines to show that a swap optimal schedule for $R2 \mid\mid C_{\max}$ instance can be as bad as $n-1$ times the optimal makespan, i.e. the performance guarantee of swap optimal schedule for $R2 \mid\mid C_{\max}$ is at least $n-1$. However, we will show that the (1+1)EA on this instance can obtain the optimal makespan in expected runtime $O(n \ln n)$.

Example I_5 [31]. There are two machines M_1 and M_2, and n jobs $J_1, \ldots, J_n$. The processing time of job J_j on machine M_i is given by

$$p_{ij} = \begin{cases} 1, & i = 1, j = 1, \ldots, n, \\ n - 1 - \frac{1}{n-1}, & i = 2, j = 1, \\ \frac{1}{n-1}, & i = 2, j = 2, \ldots, n. \end{cases}$$

In the optimal schedule with makespan one, job J_1 is assigned to machine M_1 and the other jobs are assigned to machine M_2.

Now consider the schedule in which job J_1 is assigned to M_2 and the other jobs are assigned to M_1. This schedule is swap optimal: swapping any pair of jobs from one machine to another machine does not decrease the makespan. Furthermore, it has a performance ratio of $n-1$.

Theorem 8 *Starting with any initial solution, the (1+1)EA on instance I_5 reaches the optimal schedule in expected time $O(n \ln n)$.*

Proof The proof is similar to that of Theorem 6

Since instance I_5 has two machines, again we use a bit string $x = (x_1, \cdots, x_n) \in \{0,1\}^n$ to represent a feasible solution (schedule), where $x_i = 1$ if J_i is assigned to M_1 and otherwise it is assigned to machine M_2.

The objective (fitness) function to be minimized is the makespan of the schedule. We partition the search space $\{0,1\}^n$ into three disjointed sets B_1, B_2 and B_3 with

$$B_1 = \{x \in \{0,1\}^n \mid makespan(x) > n-1\},$$
$$B_2 = \{x \in \{0,1\}^n \mid makespan(x) = n-1\},$$
$$B_3 = \{x \in \{0,1\}^n \mid makespan(x) < n-1\}.$$

If the current solution belongs to B_1, we consider two cases: the critical machine is M_1 and otherwise M_2.

If M_1 defines the makespan, then all jobs are assigned to M_1. Hence, moving any job from M_1 to M_2 decreases the fitness value by at least one, and the probability for this event is at least $\frac{1}{e}$.

If M_2 defines the makespan, then job J_1 and at least two other jobs are assigned to M_2. Hence, moving J_1 from M_2 to M_1 decreases the fitness value at least one, and again the probability for this event is at least $\frac{1}{en}$.

We conclude that the algorithm leaves B_1 in expected time $O(n)$.

Now we assume the current string belongs to B_2. We consider two cases: job J_1 is assigned to M_1 and otherwise M_2.

If J_1 is assigned to M_1, M_1 defines the makespan that equals $n-1$. It follows that moving any job except J_1 from M_1 to M_2 decreases the fitness value by at least one, and the probability for this event is at least $\frac{1}{en}$.

If J_1 is assigned to M_2, the number of other jobs that are assigned to M_1 is not less than $n-2$ since the current makspan is $n-1$. Therefore moving J_1 from M_2 to M_1 and two other jobs from M_1 to M_2 decreases the fitness value by at least one, and the probability for this event is $\Omega(\frac{1}{n})$.

We conclude that the algorithm leaves B_2 in expected time $O(n)$.

Now we assume the current string belongs to B_3. Note that job J_1 is assigned to M_1 in this phase. If the current schedule is not the optimal schedule, moving a job except J_1 from M_1 to M_2 decreases the fitness value by at least one. There are at most $n-3$ different fitness values in this phase, therefore similar to the scenario of OneMax function [7], the (1+1)EA reaches the optimal schedule in expected time $O(n \ln n)$.

Altogether the theorem follows. □

5 Conclusion

In this paper, we present the theoretical analysis of EAs for the multiprocessor scheduling problem. For $Q2 \mid\mid C_{max}$ problem, we demonstrate that the (1+1)EA can achieve an approximation ratio of $\frac{1+\sqrt{5}}{2}$ in polynomial time. We also show that EAs beat local search algorithms on three selected multiprocessor scheduling problem instances. A natural question is whether EAs can obtain the other approximation ratios in Table 1 in polynomial time. Furthermore, we may ask whether EAs can improve these approximation bounds on local search algorithms. Another interesting problem is, what is the performance of EAs on practical multiprocessor scheduling problems, such as online

scheduling problem which is common to Youtube server. There still is much work to be done to understand well how EAs work on NP-hard problems.

Acknowledgments Y. Zhou supported by National Natural Science Foundation of China under the Grant 61170081. J. Zhang supported by National Natural Science Foundation of China under the Grants 61125205 and 61332002. Y. Wang supported by National Natural Science Foundation of China under the Grant 61273314.

References

1. Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford (1996)
2. Rudolph, G.: Convergence Properties of Evolutionary Algorithms. Hamburg Kovac, Hamburg (1997)
3. Oliveto, P.S., He, J., Yao, X.: Time complexity of evolutionary algorithms for combinatorial optimization: a decade of results. Int. J. Autom. Comput. **4**(3), 281–293 (2007)
4. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity. Springer, New York (2010)
5. Karlin, S., Taylor, H.M.: A First Course in Stochastic Processes, 2nd edn. Academic Press, New York (1975)
6. Mitzenmacher, M., Upfal, E.: Propability and Computing. Cambridge University Press, Cambridge (2005)
7. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theor. Comput. Sci. **276**, 51–81 (2002)
8. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artifi. Intell. **127**(1), 57–85 (2001)
9. Giel, O., Wegener, I.: Evolutionary algorithms and the maximum matching problem. In: Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science, vol. 2607, pp. 415–426. Speringer-Verlag, Berlin (2003)
10. Neumann, F., Wegener, I.: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Theor. Comput. Sci. **378**(1), 32–40 (2007)
11. Doerr, B., Happ, E., Klein, C.: A tight analysis of the (1+1)-EA for the single source shortest path problem. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC'07), pp. 1890–1895. IEEE Press, Singapore (2007)
12. Baswana, S., Biswas, S., Doerr, B., Friedrich, T., Kurur, P.P., Neumann, F.: Computing single source shortest paths using single-objective fitness functions. In: Jansen, T., Garibay, I., Wiegand, R.P., Wu, A.S. (eds.) Proceedings of the Tenth International Workshop on Foundations of Genetic Algorithms (FOGA'09), pp. 59–66. ACM Press, Orlando (2009)
13. He, J., Yao, X.: From an individual to a population: an analysis of the first hitting time of populationbased evolutionary algorithms. IEEE Trans. Evol. Comput. **6**(5), 495–511 (2002)
14. Jansen, T., Jong, K.A.D., Wegener, I.: On the choice of the offspring population size in evolutionary algorithms. Evol. Comput. **13**(4), 413–440 (2005)
15. Chen, T., Tang, K., Chen, G., Yao, X.: A large population size can be unhelpful in evolutionary algorithms. Theor. Comput. Sci. **436**(8), 54–70 (2012)
16. Chen, T., He, J., Sun, G., Chen, G., Yao, X.: A new approach to analyzing average time complexity of population-based evolutionary algorithms on unimodal problems. IEEE Trans. Syst. Man Cybernet. B **39**(5), 1092–1106 (2009)
17. Garey, M.R., Johnson, D.S.: Computers and Intractability—A Guide to the Theory of NP-completeness. Freeman, New York (1979)
18. Witt, C.: Worst-case and average-case approximations by simple randomized search heuristics. In: Proceedings of the 22th Symposium on Theoretical Aspects of Computer Science Proceedings (STACS '05). Lecture Notes on Computer Science, vol. 3404, pp. 44–56. Springer, Heidelberg (2005)
19. Gunia, C.: On the analysis of the approximation capability of simple evolutionary algorithms for scheduling problems. In: Beyer, H.G., O'Reilly, U.M. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 571–578. ACM, New York (2005)
20. Sutton, A. M., Neumann, F.: A parameterized runtime analysis of simple evolutionary algorithms for makespan scheduling. In: Proceedings of the Twelfth International Conference on Parallel Problem

Solving from Nature (PPSN'12), Part I. Lecture Notes in Computer Science, vol. 7491, pp. 52–61. Speringer-Verlag, Berlin (2012)
21. Friedrich, T., He, J., Hebbinghaus, N., Neumann, F., Witt, C.: Analyses of simple hybrid evolutionary algorithms for the vertex cover problem. Evol. Comput. **17**(1), 3–20 (2009)
22. Yu, Y., Yao, X., Zhou, Z.H.: On the approximation ability of evolutionary optimization with application to minimum set cover. Artif. Intell. **180–181**, 20–33 (2012)
23. Sutton, A. M., Neumann, F.: A parameterized runtime analysis of evolutionary algorithms for the euclidean traveling salesperson problem. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, pp. 1105–1111. AAAI Press, Milano (2012)
24. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy, Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discr. Math. **5**, 287–326 (1979)
25. Potts, C., Strusevich, V.: Fifty years of scheduling: a survey of milestones. J. Oper. Res. Soc. **60**(S1), 41–68 (2009)
26. Finn, G., Horowitz, E.: A linear time approximation algorithm for multiprocessor scheduling. BIT **19**, 312–320 (1979)
27. Cho, Y., Sahni, S.: Bounds for list schedules on uniform processors. SIAM J. Comput. **9**(1), 91–103 (1980)
28. Langston, M.: Improved 0/1-interchange scheduling. BIT Numer. Math. **22**(3), 282–290 (1982)
29. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. J. ACM **34**, 144–162 (1987)
30. Lenstra, J.K., Shmoys, D.B., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**, 259–271 (1990)
31. Schuurman, P., Vredeveld, T.: Performance guarantees of local search for multiprocessor scheduling. Inf. J. Comput. **19**(1), 52–63 (2007)
32. Ammons, J.C., Lofgren, C.B., McGinnis, L.F.: A large scale machine loading problem in flexible assembly. Ann. Oper. Res. **3**(7), 317–332 (1985)
33. Stecke, K.E.: Design, planning, scheduling, and control problems of flexible manufacturing systems. Ann. Oper. Res. **3**(1), 1–12 (1985)
34. Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., Baruah, S.: A categorization of real-time multiprocessor scheduling problems and algorithms. In: Joseph, Y.Leung (ed.) Handbook on Scheduling Algorithms, Methods, and Models, pp. 30.1–30.19. Chapman Hall/CRC, Boca Raton (2004)
35. Michiels, W., Aarts, E., Korst, J.: Theoretical Aspects of Local Search. Springer, Berlin (2007)