Artificial Immune Systems Can Find Arbitrarily Good Approximations for the NP-Hard Partition Problem

Dogan Corus(✉), Pietro S. Oliveto, and Donya Yazdani

Rigorous Research, University of Sheffield, Sheffield, UK
{d.corus,p.oliveto,dyazdani1}@sheffield.ac.uk

Abstract. Typical Artificial Immune System (AIS) operators such as hypermutations with mutation potential and ageing allow to efficiently overcome local optima from which Evolutionary Algorithms (EAs) struggle to escape. Such behaviour has been shown for artificial example functions such as Jump, Cliff or Trap constructed especially to show difficulties that EAs may encounter during the optimisation process. However, no evidence is available indicating that similar effects may also occur in more realistic problems. In this paper we perform an analysis for the standard NP-Hard Partition problem from combinatorial optimisation and rigorously show that hypermutations and ageing allow AISs to efficiently escape from local optima where standard EAs require exponential time. As a result we prove that while EAs and Random Local Search may get trapped on 4/3 approximations, AISs find arbitrarily good approximate solutions of ratio $(1+\epsilon)$ for any constant ϵ within a time that is polynomial in the problem size and exponential only in $1/\epsilon$.

1 Introduction

Artificial Immune Systems (AIS) take inspiration from the immune system of vertebrates to solve complex computational problems. Given the role of the natural immune system to recognise and protect the organism from viruses and bacteria, natural applications of AIS have been pattern recognition, computer security, virus detection and anomaly detection [1–3]. Various AIS, inspired by Burnet's clonal selection principle, have been devised for solving optimisation problems. Amongst these, the most popular are Clonalg [4], the B-Cell algorithm [5] and Opt-IA [6].

AIS for optimisation are very similar to evolutionary algorithms (EAs) since they essentially use the same Darwinian evolutionary principles to evolve populations of solutions (here called antibodies). In particular, they use the same natural selection principles to gradually evolve high quality solutions. The main distinguishing feature of AIS to more classical EAs is their use of variation operators that typically have higher mutation rates compared to the standard bit mutations (SBM) of EAs (such as the contiguous somatic mutations of the B-Cell algorithm and the hypermutations with mutation potential of Opt-IA) and

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 16–28, 2018.
https://doi.org/10.1007/978-3-319-99259-4_2

their use of ageing operators that remove old solutions i.e., that have spent a long time without improving (local optima). Despite their popularity it is still largely unclear on what problems an AIS will have better performance to that of EAs. Also very little guidance is available on when a class of AIS should be applied rather than another. Amongst the few available results, it has been proven that there exist instance classes of both vertex cover [7] and the longest common subsequence [8] NP-Hard problems that are hard for EAs equipped with SBM and crossover for which the B-Cell algorithm is efficient The superior performance is due to the ability of the contiguous somatic mutations of the B-Cell algorithm to efficiently escape the local optima of these instances while SBM require exponential expected time in the size of the instance.

Apart from these results, the theoretical understanding of AIS relies on analyses of their behaviour for artificially constructed toy problems. Recently it has been shown how both the hypermutations with mutation potential and the ageing operator of Opt-IA can lead to considerable speed-ups compared to the performance of well-studied EAs using SBM for standard benchmark functions used in evolutionary computation such as Jump, Cliff or Trap [9]. While the performance of hypermutation operators to escape the local optima of these functions is comparable to that of the EAs with high mutation rates that have been increasingly gaining popularity since 2009 [10–14], ageing allows the optimisation of hard instances of Cliff in $O(n \log n)$, a runtime that is required by the SBM of typical EAs to optimise any function with unique optimum i.e., SBM require $\Theta(n \log n)$ to optimise its easiest function with unique optimum - OneMax [15]. Although some of these speed-ups over standard EA performance are particularly impressive, no similar evidence of superior performance of these operators is available for more realistic problems.

In this paper we perform an analysis for Partition, a classical NP-Hard combinatorial optimisation problem for which the performance of Random Local Search (RLS) and of the $(1+1)$ EA is understood [16–18]. Since Partition is essentially a makespan scheduling problem with two machines, its relevance to practical applications can be easily seen. It is well understood that both RLS and the $(1+1)$ EA may get stuck on local optima which lead to a worst case approximation ratio of $4/3$. In order to achieve a $(1+\epsilon)$ approximation for arbitrary ϵ a clever restart strategy has to be put in place. Herein, we first show the power of hypermutations and ageing by proving that each of them solve to optimality instances that are hard for RLS and SBM EAs, by efficiently escaping local optima for which the EAs struggle. Afterwards we prove that AIS using hypermutations with mutation potential guarantee arbitrarily good solutions of approximation ratio $(1+\epsilon)$ in expected time $n(\epsilon^{-(2/\epsilon)-1})(1-\epsilon)^{-2}e^3 2^{2/\epsilon} + n^3 2^{2/\epsilon} + n^3$, which reduces to $O(n^3)$ for any constant ϵ without requiring any restarts. On the other hand, we prove that an AIS with SBM and ageing can efficiently achieve the same approximation ratio in $O(n^2)$, automatically restarting the optimisation process by implicitly detecting when it is stuck on a local optimum. To the best of our knowledge this is the first time either hypermutations or ageing have been theoretically analysed for a standard problem from combinatorial optimi-

Algorithm 1. (1+1) IA^{hyp} [9] for minimisation

1: Set each bit in x to 1 with probability 1/2 and to 0 otherwise, then evaluate $f(x)$.
2: **while** termination condition not satisfied **do**
3: $y := x$; $Flip := \{1, \ldots, n\}$;
4: **while** $Flip \neq \emptyset$ and $f(y) \geq f(x)$ **do**
5: $i :=$ Sample $Flip$ u. a. r.; $Flip := Flip \setminus i$; flip y_i; evaluate $f(y)$;
6: **end while**
7: If $f(y) \leq f(x)$, then $x := y$.
8: **end while**

sation and the first time performance guarantees of any AIS are proven in such a setting.

Due to space limitations some proofs are omitted for this extended abstract[1].

2 Preliminaries

Hypermutation with mutation potential operators are inspired by the high mutation rates occurring in the natural immune system [6]. For the purpose of optimisation, these high mutation rates may allow the algorithm to escape local optima by identifying promising search areas far away from the current ones.

In this paper we will analyse the static hypermutation operator considered in [9] for benchmark functions, where the maximum number of bits to be flipped is fixed to $M = cn$ throughout the optimisation process. Two variants of static hypermutations have been proposed in the literature. A straightforward version, where in each mutation exactly cn bits are flipped, and another one called *stop at first constructive mutation* (FCM) where the solution quality is evaluated after each of the cn bit-flips and the operator is halted once a *constructive mutation* occurs. Since Corus et al. [9] proved that the straightforward version requires exponential time to optimise any function with a polynomial number of optima, we will consider the version with FCM. We define a mutation to be *constructive* if the solution is strictly better than the original parent and we set $c = 1$ such that all bits will flip if no constructive mutation is found before. For the sake of understanding the potentiality of the operator, we embed it into a minimal AIS framework that uses only one antibody (or individual) and creates a new one in each iteration via hypermutation as done previously in the literature [9]. The algorithm is essentially a $(1 + 1)$ EA [20,21] that uses hypermutations instead of SBM. The simple AIS for the minimisation of objective functions, called $(1 + 1)$ IA^{hyp} for consistency with the literature, is formally described in Algorithm 1.

Another popular operator used in AIS is *Ageing*. The idea behind the operator is to remove antibodies which have not improved for a long time. Intuitively, these antibodies are not improving because they are trapped on some local optimum, and they may be obstructing the algorithm from progressing in more promising areas of the search space (i.e., the population of antibodies may

[1] A complete version of the paper including all the proofs is available on arXiv [19].

Algorithm 2. $(\mu + 1)$ EA^{ageing} [22] for minimisation

1: Create population $P := \{x_1, \cdots, x_\mu\}$ with each bit in x_i set to 1 with probability $1/2$ and to 0 otherwise;
2: For all $x \in P$ evaluate $f(x)$ and set $x[age] := 0$.
3: **while** termination condition not satisfied **do**
4: For all $x \in P$ set $x[age] := x[age] + 1$.
5: Select $x \in P$ uniformly at random;
6: $y := x$ and flip each bit in y with probability $1/n$. Add y to P.
7: If $f(y) < f(x)$, then $y[age] := 0$. Else, $y[age] := x[age]$;
8: For all $x \in P$ if $x[age] \geq \tau$ then remove x from P;
9: If $|P| > \mu$ then remove the individual from P with the highest $f(x)$;
10: If $|P| < \mu$ then add enough number of randomly created individuals to P until $|P| = \mu$;
11: **end while**

quickly be taken over by a high quality antibody on a local optimum). The antibodies that have been removed by the ageing operator are replaced by new ones initialised at random.

Ageing operators have been proven to be very effective at automatically restarting the AIS, without having to set up a restart strategy in advance, once it has converged on a local optimum [23]. Stochastic versions have been shown to also allow antibodies to escape from local optima [9,22]. As in previous analyses we incorporate the ageing operator in a simple $(\mu+1)$ EA algorithmic framework and for simplicity consider the static variant where antibodies are removed from the population with probability 1 if they have not improved for τ generations. The algorithm is formally defined in Algorithm 2. We will compare the performance of the AIS with the standard $(1 + 1)$ EA [21,24] and RLS for which the performance for Partition is known. The former uses standard bit mutation i.e., it flips each bit of the parent with probability $1/n$ in each iteration, while the latter flips exactly one bit.

Given n jobs with processing times $p_1, \ldots, p_n$ and $p_i > 0$, the Partition problem is that of scheduling the jobs on two identical machines, M_1 and M_2, in a way that the overall completion time (i.e., the *makespan*) is minimised. This simple to define scheduling problem is well studied in theoretical computer science and is known to be NP-Hard [25]. Hence, it cannot be expected that any algorithm finds exact solutions to all instances in polynomial time. However, there exist efficient problem specific algorithms which guarantee solutions with approximation ratio $(1 + \epsilon)$ in time $O(n^3/\epsilon)$, in classical complexity measures, which is polynomial both in n and $1/\epsilon$ [26]. Let f_A be the solution quality guaranteed by algorithm A and f_{opt} be the value of the optimal solution. Then the approximation ratio for a minimisation problem is defined as f_A/f_{opt}.

For the application of randomised search heuristics a solution may easily be represented with a bitstring $x \in \{0,1\}^n$ where each bit i represents job i and if the bit is set to 0 it is assigned to the first machine (M_1) and otherwise to the second machine (M_2). Hence, the goal is to minimise

$f(x) := \max\left\{\sum_{i=1}^{n} p_i x_i, \sum_{i=1}^{n} p_i(1-x_i)\right\}$, i.e., the processing time of the last machine to terminate. Both RLS and the (1 + 1) EA have 4/3 worst case expected approximation ratios for the problem (i.e. there exist instances where they can get stuck on solutions by a factor of 4/3 worse than the optimal one) [16,17]. However if an appropriate restart strategy is setup in advance, both algorithms may be made into polynomial randomised approximation schemes (PRAS, i.e., algorithms that compute a $(1+\epsilon)$ approximation in polynomial time in the problem size with probability at least 3/4) with a runtime bounded by $O(n\ln(1/\epsilon)) \cdot 2^{(e\log e+e)\lceil 2/\epsilon\rceil \ln(4/\epsilon)+O(1/\epsilon)}$ [16,17]. Hence, as long as ϵ does not depend on the problem size, the algorithms can achieve arbitrarily good approximations with an appropriate restart strategy.

In this paper we will show that AIS can achieve stronger results. Firstly, we will show that both ageing and hypermutations can efficiently solve to optimality the worst-case instances for RLS and the (1 + 1) EA. More importantly, we will prove that ageing automatically achieves the necessary restart strategy to guarantee the $(1+\epsilon)$ approximation while hypermutations guarantee it in a single run in polynomial expected runtime and with overwhelming probability.

3 Generalised Worst-Case Instance

The instance from the literature P^*_ϵ leading to a 4/3 worst-case expected approximation ratio for RLS and the (1 + 1) EA consists of two large jobs with long processing times $p_1 := p_2 := 1/3 - \epsilon/4$ and the remaining $n-2$ jobs with short processing times $p_i := (1/3+\epsilon/2)/(n-2)$ for $3 \leq i \leq n$ (the sum of the processing times are normalised to 1 for cosmetic reasons) [16]. Any partition where one large job and half of the small jobs are placed on each machine is naturally a global optimum of the instance (i.e., the makespan is 1/2). Note that the sum of all the processing times of the small jobs is slightly larger than the processing time of a large job. The local optimum leading to the 4/3 expected approximation consists of the two large jobs on one machine and all the small jobs on the other. The makespan of the local optimum is $p_1 + p_2 = 2/3 - \epsilon/2$ and, in order to decrease it, a large job has to be moved to the fuller machine and at the same time at least $\Omega(n)$ small jobs have to be moved to the emptier machine. Since this requires to flip $\Omega(n)$ bits, which cannot happen with RLS and only happens with exponentially small probability $n^{-\Omega(n)}$ with the SBM of EAs, their expected runtime to escape from such a configuration is at least exponential.

In this section, to highlight the power of the AIS to overcome hard local optima for EAs, we generalise the instance P^*_ϵ to contain an arbitrary number $s = \Theta(1)$ of large jobs and show how the considered AIS efficiently solve the instance class to optimality by escaping from local optima efficiently while RLS and the (1 + 1) EA cannot. Hence, hypermutations and ageing are efficient on a vast number of instances where SBM and local mutations alone fail. The generalised instance class G^*_ϵ is defined as follows.

Definition 1. *The class of* PARTITION *instances G^*_ϵ are characterised by an even number of jobs n, an even number of large jobs $s = \Theta(1)$ and the following processing times:* $p_i = \begin{cases} \frac{1}{2s-1} - \frac{\epsilon}{2s} & \text{if } i \leq s \\ \frac{s-1}{n-s} \cdot (\frac{1}{2s-1} + \frac{\epsilon}{2(s-1)}) & \text{otherwise} \end{cases}$ *where $0 < \epsilon < 1/(2s-1)$ is an arbitrarily small constant.*

The instance class has the same property as the instance P^*_ϵ. If all the large jobs are placed on one machine and all the small jobs on the other, then at least $\Omega(n)$ small jobs need to be moved in exchange for a large job to decrease the discrepancy (the difference between the loads of the machines). Such a local optimum allows to derive a lower bound on the worst-case expected approximation ratio of algorithms that get stuck there. Obviously the greater the number of large jobs, the smaller the bound on the approximation ratio will be. We now prove that the $(1+1)$ EA has exponential expected runtime on G^*_ϵ. The proof is similar to that of the 4/3 approximation [16]. It essentially shows that with constant probability the algorithm gets trapped on the local optimum. Then the statement will follow by showing that exponential expected time is required to escape from there. That RLS also fails is essentially a corollary.

Theorem 1. *The $(1+1)$ EA needs at least $n^{\Omega(n)}$ fitness function evaluations in expectation to optimise any instance of G^*_ϵ.*

In the following subsection we will show that hypermutations are efficient. Afterwards we will do the same for ageing.

3.1 Hypermutations

We will start this subsection by proving some general statements about hypermutations. The hypermutation operator flips all the bits in a bitstring successively and evaluates the fitness after each step. Unless an improvement is found first, each hypermutation operation results in a sequence of n solutions sampled in the search space. This sequence of solutions contains exactly one bitstring for each Hamming distance $d \in [n]$ to the initial bitstring. Moreover, the string which will be sampled at distance d is uniformly distributed among all the bitstrings of distance d to the input solution. Thus, as we approach the middle, the number of possible outcomes grows exponentially large but the first and the last few strings sampled are picked among a polynomially large subset of the search space. We will now provide two lemmata regarding the probability of particular outcomes in the first and the last m bitstrings of the sequence.

Lemma 1. *Given that no improvement is found, the probability that k specific bit positions are flipped by the hypermutation operator in the first (or last) $m \geq k$ mutation steps is at least $\left(\frac{m-k+1}{n-k+1}\right)^k$.*

Lemma 2. *Let x^i be the ith bitstring sampled by the hypermutation and s^i the substring of x^i which consists of bit positions $S \subset [n]$. For any given target*

substring s^* *and integer* $m > |S|$*, the probability that* $\forall i \in \{m, m+1, \ldots, n-m-1, n-m\}$*,* $s^i = s^*$ *is at least* $\left(\frac{m-|S|+1}{n-|S|+1}\right)^{|S|}$.

We can observe that the probability bounds we get from these lemmata are polynomial if k (or $|S|$) is a constant. Moreover, if both $k = \Theta(1)$ and $m = \Omega(n)$ we obtain probabilities in the order of $\Omega(1)$.

For G^*_ϵ, we would like to distribute two kinds of jobs evenly among machines. While one type of job is constant in number, the other is in the order of $\Theta(n)$. So the previous lemmata would provide reasonable probability bounds only for the large jobs. For the small jobs we will make use of the fact that the configuration we want is not any configuration, but an exact half and half split. Intuitively, it is obvious that if all the jobs start on one machine, at exactly the $n/2$th bit flip, the split will be half and half. However, as the starting distribution gets further away from the extremes, it becomes less clear when the split will exactly happen. Fortunately, the fact that the number of small jobs is large, will work in our favor to predict the time of the split more precisely. Whilst the previous lemmata provide bounds for the first and last bitflips of hypermutation, we will use Serfling's concentration bound [27] on hypergeometric distributions to prove the following theorem about the bitflips in the middle.

Theorem 2. *If the input bitstring of hypermutation has* $\left(\frac{1}{2}+a\right)n$ *1-bits for some constant* $a > 0$*, then the probability that any solution sampled after the* $n\frac{a}{2a-c}$*th mutation step for any* $a > c > 0$ *to have more than n/2 1-bits is in the order of* $e^{-\Omega(nc^2)}$.

Corollary 1. *If the input bitstring of hypermutation has* $\left(\frac{1}{2}+a\right)n$ *1-bits for some constant* $a > 0$*, then with probability* $1 - e^{-\Omega(nc^2)}$*, there exists a* $k \in \{n\frac{a-c}{2a-c}, \ldots, n\frac{a}{2a-c}\}$ *for any positive* $a > c = \omega(1/\sqrt{n})$*, such that the number of 1-bits in the kth solution sampled by hypermutation has exactly n/2 1-bits.*

Now we have all the necessary tools to prove that the $(1+1)$ IA^{hyp} can solve G^*_ϵ efficiently. The heart of the proof of the following theorem is to show that from any local optimum, hypermutations identify the global optimum with constant probability unless an improvement is found first. The proof then follows because there are at most $O(n)$ different fitness levels.

Theorem 3. *The* $(1+1)$ IA^{hyp} *optimises the* G^*_ϵ *class of instances in* $O(n^2)$ *expected function evaluations.*

Since the worst case instance for the $(1+1)$ EA [16] is an instance of G^*_ϵ with $s = 2$, the following corollary holds.

Corollary 2. *The* $(1+1)$ IA^{hyp} *optimises* P^*_ϵ *in* $O(n^2)$ *expected function evaluations.*

3.2 Ageing

In this section we will show that the $(\mu + 1)$ EA^{ageing} can optimise the G^*_ϵ instances efficiently. Our approach to prove the following theorem is to first show that if a solution where the large jobs are equally distributed is sampled in the initialisation, then it takes over the population and the $(\mu + 1)$ EA^{ageing} quickly finds the optimum. The contribution of ageing is considered afterwards to handle the case when no such initial solution is sampled. Then, we will show that whenever the algorithm gets stuck at a local optima, it will reinitialise the whole population after τ iterations and sample a new population.

Theorem 4. *The* $(\mu + 1)$ *EA*^{ageing} *optimises the* G^*_ϵ *class of instances in* $O(\mu n^2 + \tau)$ *steps in expectation for* $\tau = \Omega(n^2)$ *and* $\mu = O(\log n)$.

Clearly the following corollary holds as P^*_ϵ is an instance of G^*_ϵ.

Corollary 3. *The* $(\mu+1)$ *EA*^{ageing} *optimises* P^*_ϵ *in* $O(\mu n^2+\tau)$ *steps in expectation for* $\tau = \Omega(n^2)$ *and* $\mu = O(\log n)$.

4 $(1 + \epsilon)$ Approximation Ratios

4.1 Hypermutations

In the next theorem we will show that the $(1 + 1)$ IA^{hyp} can efficiently find arbitrarily good constant approximations to any PARTITION instance. Before we state our main theorem, we will require the following helper lemma.

Lemma 3. *Let* x^i *be the ith bitstring sampled by the hypermutation,* s^i *the substring of* x^i *which consists of bit positions* $S \subset [n]$*, and,* $f(x) := \sum_{j \in S} x_j w_j$ *a linear function defined on the substring for some non-negative weights* w_j*. Given that the input bitstring of the hypermutation is* 0^n*, the expected value of* $f(x^i)$ *is* $\frac{i}{n} \sum_{j \in S} w_j$.

In the proof of the following theorem we first divide the search space into $2^{2/\epsilon}$ subspaces according to the distribution of the largest $2/\epsilon$ jobs and then further divide each subspace into the same n fitness levels used for the proof of the $(1 + 1)$ EA in [16]. However, we show that in each fitness level, hypermutations have a good probability of either finding a $(1 + \epsilon)$ approximation or leaving the fitness level for good. The statement then follows by summing over the $n2^{2/\epsilon}$ fitness levels.

Theorem 5. *The* $(1+1)$ *IA*^{hyp} *finds a* $(1+\epsilon)$ *approximation to any instance of* PARTITION *in at most* $n(\epsilon^{-(2/\epsilon)-1})(1-\epsilon)^{-2}e^3 2^{2/\epsilon} + n^3 2^{2/\epsilon} + n^3$ *fitness function evaluations in expectation for any* $\epsilon = \omega(n^{-1/2})$.

Proof. In order to prove our upper bound on the runtime, we will divide the run of the algorithm into at most $2^{2/\epsilon}$ phases and find a bound on the expected time that is valid for all phases.

Following the proof of Theorem 3 in [16], we refer to the $s := \lceil 2/\epsilon \rceil - 1 \leq n/2$ jobs with the largest processing times as *large jobs* and to the rest as the *small jobs*. Let P be the sum of the processing times of all jobs. Since $p_1 \geq p_2 \geq \ldots \geq p_n$ w.l.o.g., we know that $p_i \leq \epsilon P/2$ for all $i > s$ because otherwise the sum of the first $s+1$ jobs would be larger than P.

Consider the 2^s partitions of only the large objects. We sort these 2^s partitions according to non-increasing makespan and denote the ith partition in the sequence as y_i. Now, we can divide the solution space of the original problem into subspaces A_i for $i \in [2^s]$, where A_i consists of all the solutions where the large jobs are distributed as in y_i. Let x_i^* denote the solution with the best fitness value in A_i.

Let $d(x)$ denotes the discrepancy of solution x. We define k as the smallest index i that satisfies $d(y_i) \leq \sum_{j=s+1}^{n} p_j$. Thus, for any large jobs configuration y_j for $j < k$, if all the small jobs are assigned to the emptier machine of y_j, then the load of the obtained solution is the same as the load of y_j itself because the discrepancy is larger than the sum of the processing times of the small jobs by definition. Since the makespan of y_j is a lower bound on the makespan of any solution $x \in A_j$, the solution where all small jobs are assigned to the emptier machine of y_i is the best solution in A_i for all $i < k$. Again for $i < k$, we can say that once the $(1+1)$ IA^{hyp} finds a solution with better fitness value than x_i^*, any solution that belongs to any set A_j for $j \leq i$ will be rejected by the algorithm due to its inferior fitness value. This allows us to divide the time until a solution better than x_{k-1}^* is found for the first time into $k-1$ distinct phases where during phase i the fitness of the current solution is between $f(x_{i-1}^*)$ and $f(x_i^*)$ (we define $f(x_0^*) := P$ for completeness). We will now further divide the expected length of phase i, into the expected time until x_i^* is found given an arbitrary solution x that satisfies $f(x_{i-1}^*) > f(x) > f(x_i^*)$, and the expected time until an improvement is found given that the current solution is $f(x_i^*)$.

We start with the expected time until x_i^* is found given an initial solution x such that $f(x_{i-1}^*) > f(x) > f(x_i^*)$ holds. Since $f(x_{i-1}^*) > f(x)$, the makespan of the underlying large job configuration of x is at least as good as the makespan of y_i and thus the makespan of x can be upper bounded by the load obtained when all small jobs are assigned to the fuller machine of y_i. Since $i < k$ we also know that the fuller machine of x_i^* has no small jobs on it. Thus, during phase i we can bound the fitness in the interval $[f(y_i), f(y_i) + \sum_{j=s+1}^{n} p_j]$. We further divide this interval into the following levels, $L_\ell := \left\{ x \mid f(y_i) + \sum_{j=s+\ell}^{n} p_j \geq f(x) > f(y_i) + \sum_{j=s+\ell+1}^{n} p_j \right\}$, implying that for any solution at level L_ℓ there is at least one job with processing time at least $p_{s+\ell}$ which can be moved to the emptier machine with probability $1/n$ as the first bit-flip and yield a solution at level $L_{\ell+1}$. Since there are at most n levels for the phase i, the expected number of iterations until the level L_{n-s+1}, which contains the solution x_i^* is discovered is at most n^2.

Secondly, we will bound the expected time until an improvement is found given that the current solution is $f(x_i^*)$. In order to do this we will bound the probability that a $(1+\epsilon)$ approximation is obtained in a single iteration given that no other improvements are found in the previous mutation steps of the hypermutation operator by some value $p_{\approx}$. This will allow us to claim that in expected $p_{\approx}^{-1}$ generations we will either find the approximation we sought or at least leave the subspace A_i for good.

Consider the optimal configuration of large jobs y_{2^s} and denote its makespan as L. Since both y_{2^s} and its complementary bitstring have the same makespan, w.l.o.g., we will assume the fuller machines of y_i and y_{2^s} are both M_1. According to Lemma 2, the $s \leq 2/\epsilon$ large jobs are assigned identically to y_{2^s} between the $n(\epsilon-\epsilon^2)$th and $n-n(\epsilon-\epsilon^2)$th bit-flips with probability at least $\left(\epsilon-\epsilon^2\right)^{2/\epsilon} e^{-1}$ Since the initial solution x_i^* does not assign any small jobs to M_1 and the sum of the processing times of the small jobs is less than $P/2$, by the $n(\epsilon-\epsilon^2)$th bit-flip the expected total processing time of small jobs moved from M_2 to M_1 is at most $(\epsilon-\epsilon^2)P/2$ according to Lemma 3. Due to Markov's inequality, with probability at least $1-((\epsilon-\epsilon^2)P/(\epsilon)P) = \epsilon$ the moved sum is less than $\epsilon P/2$ and the makespan of the solution is at most $L+\epsilon P/2$. In general, $OPT \geq L$ with OPT indicating the optimal makespan, since introducing the small jobs cannot improve the makespan and also $OPT \geq P/2$ since a perfect split is the best possible partition. Thus $(L+\epsilon P/2)/OPT$ is less than $(1+\epsilon)$. This implies that with probability

$$p_{\approx} \geq \frac{(\epsilon-\epsilon^2)^{2/\epsilon}}{e}\epsilon = \frac{\epsilon^{(2/\epsilon)+1}}{e(1-\epsilon)^{-2/\epsilon}} = \frac{\epsilon^{(2/\epsilon)+1}(1-\epsilon)^2}{e(1-\epsilon)^{-((1/\epsilon)-1)2}} \geq \frac{\epsilon^{(2/\epsilon)+1}(1-\epsilon)^2}{e^3}$$

a $(1+\epsilon)$ approximation is found unless an improvement is obtained before. The total expected waiting time to observe either an approximation or an improvement in all local optima y_i for $i<k$ is at most $(1/\epsilon^{(2/\epsilon)+1})(1-\epsilon)^{-2}e^3 2^{2/\epsilon}$, since $k \leq 2^s \leq 2^{2/\epsilon}$. If we add the time spent in between local optima, $n^2 2^{2/\epsilon}$, we obtain the bound on the expected number of iterations until subspace A_k is reached for the first time.

Once subspace A_k is reached, by definition the underlying large job configuration has a discrepancy which is not large enough to fit all small jobs. This means that when the next local optimum is found, the discrepancy is less than half of the processing time of a small job. Thus, the locally optimal solution is a $(1+\epsilon)$ approximation since the processing time of small jobs is at most $\epsilon P/2$. Such a solution is found in n^2 expected time after the first solution in A_k is sampled. Summing up all the expected times bounded above, the expected runtime we obtain is: $(\epsilon^{-(2/\epsilon)-1})(1-\epsilon)^{-2}e^3 2^{2/\epsilon} + n^2 2^{2/\epsilon} + n^2$. □

4.2 Ageing

The following theorem shows that the $(1+1)$ EA^{ageing} can find $(1+\epsilon)$ approximations. The proof follows the same ideas used to prove that RLS and the $(1+1)$ EA achieve a $(1+\epsilon)$ approximation if an appropriate restart strategy

is put in place [16,17]. In particular, the main proof idea is to use the success probability of the simple $(1+1)$ EA and show that ageing automatically causes restarts whenever the $(1+1)$ EA fails to find the approximation. Hence, a restart strategy is not necessary for the $(1+1)$ EA^{ageing} to achieve the desired approximation.

Theorem 6. *Let* $\tau = \Omega(n^{1+c})$ *where* $c = \Omega(1)$. *The* $(1+1)$ EA^{ageing} *finds a* $(1+\epsilon)$ *approximation to any instance of* Partition *in at most* $(en^2+\tau)2^{(e\log e+e)\lceil 2/\epsilon\rceil \ln(4/\epsilon)+\lceil 4/\epsilon\rceil-1}$ *fitness function evaluations in expectation for any* $\epsilon \geq 4/n$.

5 Conclusion

To the best of our knowledge this is the first time that polynomial expected runtime guarantees of solution quality have been provided concerning AIS for a classical combinatorial optimisation problem. We presented a class of instances of Partition to illustrate how hypermutations and ageing can efficiently escape from local optima where the standard bit mutations used by EAs get stuck for exponential time. Then we showed how this capability allows the AIS to achieve arbitrarily good $(1+\epsilon)$ approximations to any instance of Partition in polynomial time for any constant ϵ. In contrast to standard EAs and RLS, that require parallel runs or restart schemes to achieve such approximations, the AIS find them in a single run. The result is achieved in different ways. The ageing operator locates more promising basins of attraction by restarting the optimisation process after implicitly detecting it has found a local optimum. Hypermutations find improved approximate solutions efficiently by performing large jumps in the search space. Naturally, the proof would also apply to the complete Opt-IA [6,9] if the ageing parameter is set large enough, i.e., $\tau = \omega(n\epsilon^{-2/\epsilon})$.

References

1. Forrest, S., Perelson, A.S., Allen, L., Cherukuri, R.: Self-nonself discrimination in a computer. In: Proceedings of 1994 IEEE Symposium on Security and Privacy, pp. 202–212 (1994)
2. Hedberg, S.: Combating computer viruses: IBM's new computer immune system. IEEE Par. Dist. Tech.: Syst. Appl. **4**(2), 9–11 (1996)
3. Dasgupta, D., Majumdar, N.S.: Anomaly detection in multidimensional data using negative selection algorithm. In: Proceedings of CEC 2002, pp. 1039–1044 (2002)
4. de Castro, L.N., Von Zuben, F.J.: Learning and optimization using the clonal selection principle. IEEE Trans. Evol. Comp. **6**(3), 239–251 (2002)
5. Kelsey, J., Timmis, J.: Immune inspired somatic contiguous hypermutation for function optimisation. In: Cantú-Paz, E. (ed.) GECCO 2003. LNCS, vol. 2723, pp. 207–218. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45105-6_26
6. Cutello, V., Nicosia, G., Pavone, M., Timmis, J.: An immune algorithm for protein structure prediction on lattice models. IEEE Trans. Evol. Comp. **11**(1), 101–117 (2007)

7. Jansen, T., Oliveto, P.S., Zarges, C.: On the analysis of the immune-inspired B-cell algorithm for the vertex cover problem. In: Liò, P., Nicosia, G., Stibor, T. (eds.) ICARIS 2011. LNCS, vol. 6825, pp. 117–131. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22371-6_13
8. Jansen, T., Zarges, C.: Computing longest common subsequences with the B-cell algorithm. In: Coello Coello, C.A., Greensmith, J., Krasnogor, N., Liò, P., Nicosia, G., Pavone, M. (eds.) ICARIS 2012. LNCS, vol. 7597, pp. 111–124. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33757-4_9
9. Corus, D., Oliveto, P.S., Yazdani, D.: On the runtime analysis of the Opt-IA artificial immune system. In: Proceedings of GECCO 2017, pp. 83–90 (2017)
10. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Proceedings of GECCO 2017, pp. 777–784 (2017)
11. Oliveto, P.S., Lehre, P.K., Neumann, F.: Theoretical analysis of rank-based mutation-combining exploration and exploitation. In: Proceedings of CEC 2009, pp. 1455–1462 (2009)
12. Corus, D., Oliveto, P.S.: Standard steady state genetic algorithms can hillclimb faster than mutation-only evolutionary algorithms. IEEE Trans. Evol. Comp. (2017)
13. Dang, D.-C., et al.: Emergence of diversity and its benefits for crossover in genetic algorithms. IEEE Trans. Evol. Comp. (2017, to appear)
14. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing new genetic algorithms. Theor. Comp. Sci. **567**, 87–104 (2015)
15. Corus, D., He, J., Jansen, T., Oliveto, P.S., Sudholt, D., Zarges, C.: On easiest functions for mutation operators in bio-inspired optimisation. Algorithmica **78**(2), 714–740 (2016)
16. Witt, C.: Worst-case and average-case approximations by simple randomized search heuristics. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 44–56. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31856-9_4
17. Neumann, F., Witt, C.: Bioinspired Computation in Combinatorial Optimization. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16544-3
18. Neumann, F., Witt, C.: On the runtime of randomized local search and simple evolutionary algorithms for dynamic makespan scheduling. In: Proceedings of the 24th International Conference on Artificial Intelligence, pp. 3742–3748. AAAI Press (2015)
19. Corus, D., Oliveto, P.S., Yazdani, D.: Artificial immune systems can find arbitrarily good approximations for the NP-Hard partition problem. arXiv e-prints (2018). http://arxiv.org/abs/1806.00300
20. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1 + 1) evolutionary algorithm. Theor. Comp. Sci. **276**(1–2), 51–81 (2002)
21. Oliveto, P.S., Yao, X.: Runtime analysis of evolutionary algorithms for discrete optimisation. In: Auger, A., Doerr, B. (eds.) Theory of Randomized Search Heuristics: Foundations and Recent Developments, chap. 2, pp. 21–52. World Scientific (2011)
22. Oliveto, P.S., Sudholt, D.: On the runtime analysis of stochastic ageing mechanisms. In: Proceedings of GECCO 2014, pp. 113–120 (2014)
23. Jansen, T., Zarges, C.: On the role of age diversity for effective aging operators. Evol. Intell. **4**(2), 99–125 (2011)
24. Lehre, P.K., Oliveto, P.S.: Theoretical analysis of stochastic search algorithms. In: Marti, R., Pardalos, P., Resende, M. (eds.) Handbook of Heuristics, pp. 1–36. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-07153-4_35-1

25. Graham, R.: Bounds on multiprocessing timing anomalies. SIAM J. App. Maths **17**, 263–269 (1969)
26. Hochbaum, D.: Appromixation Algorithms for NP-Hard Problems. PWS Publishing Company, Boston (1997)
27. Serfling, R.J.: Probability inequalities for the sum in sampling without replacement. Ann. Stat. 39–48 (1974)