Multi-Armed Bandits for MPSoC design space exploration

Calvin Ma

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

August 2016

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master's of Engineering.

Abstract

Reliability in integrated circuits is becoming a prominent issue with the miniaturization of electronics. Smaller process technologies have led to higher power densities, resulting in higher temperatures and earlier device wear-out. One way to mitigate failure is by over-provisioning resources and remapping tasks from failed components to components with spare capacity, or slack. Since the slack allocation design space is large, finding the optimal is difficult as brute-force approaches are impractical. Device lifetimes are typically evaluated using Monte-Carlo Simulation (MCS) by sampling each design equally; this method is inefficient since poor designs are evaluated as accurately as good designs. A better method will focus sampling time on the designs that are difficult to distinguish; this can be accomplished using Multi-armed Bandit (MAB) Algorithms. This work demonstrates that MAB achieves the same level of accuracy as MCS in 1.45x-5.26x fewer samples for lifetime distributions. MAB is applied as a selection algorithm in a genetic algorithm (GA). In the best case benchmark, a design with a higher lifetime of 0.01 years is found using MAB over MCS. The GA is extended to solve the multi-objective cost-lifetime problem; it was found that MAB does not offer a clear advantage. The practical significance of GA using MAB are limited.

Abstract French

La fiabilité des circuit intégrés devient un problème prédominant avec la miniaturisation des appareils électroniques. Les technologies de procédés plus petites causent des densités de puissance accrues, entranant des températures plus élevées et une usure d'appareil plus précoce. Un façon de prévenir les défaillances consiste à sur approvisionner les ressources et de réorganiser les tâches des composantes défectueux ayant des capacités non utilisées ou résiduelles. Étant donné que l'espace résiduel d'allocation conçu est grand, trouver la solution optimale est difficile car les approches force brute sont peu pratiques. Les durées de vie des appareils sont typiquement évaluées à l'aide de la Simulation Monte-Carlo (MCS) en échantillonnant chaque modèle également; cette méthode est inefficace puisque des modèles mal conçus sont évalués avec précision en tant que les modèles bien conçus. Une meilleure méthode consacrera le temps d'échantillonnage sur les modèles qui sont difficiles à différencier; ceci peut être accompli à l'aide d'algorithmes Multi-Armed Bandit (MAB). Ce travail démontre que l'algorithme MAB atteint une précision similaire à la simulation MCS en utilisant 1.45-5.26 fois moins d'échantillons. Le MAB est aussi utilisé comme un algorithme de sélection dans un algorithme génétique (GA). Dans le meilleur cas de référence, un modèle avec une durée de vie plus longue de 0,01 ans peut être trouvé par le MAB au lieu du MCS. L'algorithme GA est également appliqué pour résoudre le problème à objectif multiple d'optimisation de coût et vie. Il a été établi que l'algorithme MAB n'offre pas d'avantage évident. L'importance pratique de l'utilisation de l'algorithme GA en utilisant le MAB est limitée.

Acknowledgments

I would like to express my gratitude towards my supervisors Aditya Mahajan and Brett Meyer for their patience, support and expertise throughout my research. I would also like to thank the Faculty of Engineering for their paid assistance from the Chwang-Seto Faculty Scholar award as well as Canada Foundation for Innovation for their computing resources. Finally I would like to thank my family, friends, and colleagues for their support and advice that made my master's degree a worthwhile experience.

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Outline . . . 4

2 Related Work **5**
2.1 Lifetime-Aware design in MPSoCs . . . 5
2.2 Multi-Armed Bandits . . . 9
2.3 Contribution . . . 10

3 Methodology – Lifetime Modeling **11**
3.1 Failure Mechanisms Models . . . 11
3.2 Failure Model for Component lifetime . . . 15
3.2.1 Exponential Model . . . 16
3.2.2 Log-normal . . . 17
3.2.3 Weibull . . . 17
3.2.4 Adjusting distributions post-failure . . . 18
3.2.5 Choosing A Model . . . 19
3.3 System Lifetime . . . 19
3.3.1 System Lifetime . . . 20
3.3.2 Workload . . . 22
3.3.3 Architecture . . . 22
3.3.4 Components . . . 23

4 Multi-Armed Bandit **26**
4.1 Uniform sampling . . . 27
4.2 Sucessive Accept Reject (SAR) . . . 28
4.3 GapE . . . 30
4.4 Comparison of algorithms . . . 32
4.5 Experimental Setup . . . 35
4.5.1 Metrics . . . 36
4.6 Results and Discussion . . . 37

4.6.1 MWD3S . . . 42
4.6.2 MWD4S . . . 43
4.6.3 MPEG4S . . . 44
4.6.4 MPEG5S . . . 45
4.6.5 Discussion . . . 46
4.7 Future Work . . . 49

5 MAB in Genetic Algorithms 51
5.1 Methodology . . . 52
5.1.1 Representation . . . 53
5.1.2 Parent Selection . . . 53
5.1.3 Recombination . . . 55
5.1.4 Mutation . . . 56
5.1.5 Survivor Selection . . . 56
5.2 Experimental Setup . . . 57
5.3 Results . . . 59
5.3.1 MWD3S . . . 59
5.3.2 MWD4S . . . 60
5.3.3 MPEG4S . . . 61
5.3.4 MPEG5S . . . 62
5.3.5 Discussion . . . 62
5.4 Extension . . . 65
5.5 Future Work . . . 68

6 Conclusion 73

References 75

Chapter 1

Introduction

As innovations in new process technology continue to push the boundaries of transistor miniaturization, reliability issues are growing in prominence. Smaller process technologies lead to higher power densities, which in turn lead to temperature increases. For tightly integrated systems such as Multi-Processor System on Chips (MPSoC), this results in faster device degradation and to earlier wear-out related failures [1][2][3]. In one study, a 3x reduction in lifetime was observed when scaling from 180nm to 65nm [3].

In high reliability applications where recall of the device for repair would be impractical or otherwise expensive, redundancy is necessary to prolong device lifetime. Determining the optimal allocation of redundancy is a non-trivial problem and Multi-Armed Bandit algorithms are proposed as an optimization technique to facilitate this design process.

1.1 Background

There are two ways to implement redundancy into a system: with redundant components or with component slack.

When components are duplicated, they can be left unused until it is necessary. Redundant components can also be configured to operate simultaneously such as with Dual Modular redundant and Triple Modular redundant systems; these systems can also be operated in a degraded state after single failures to extend lifetime [4]. Although the use of spare components has been widely studied and is effective, they are more costly; allocating slack is a better alternative for cost-sensitive applications [5].

Slack refers to the over-provisioned capacity of a component, this spare overhead can be used to run tasks from failed components. The procedure of upgrading an existing design to one with slack involves a selection process where a component is swapped with another of higher capacity. In a typical MPSoC, a system will have multiple components each with a number of upgrade options. The number of design permutations that exist grows exponentially with the number of components and upgrade options. Consider a system with 7 processors each with 3 upgrade options, and 4 memories each with 9 upgrade options, the total number of designs is $7^3 \times 4^9 \approx 9 \times 10^{10}$. Although some designs will appear identical to others by specifications since they are permutations of each other, they are not necessarily equivalent ($\{Processor1, Processor2\} \neq \{Processor2, Processor1\}$). The actual lifetime of the design will depend on factors such as the floor-planning and the location of the component relative to the network switches. Overall, since the bandwidth load on the network switches may be different in each design, it is possible for one design to have a better lifetime than another even though they appear identical. This adds to the challenge of finding the best designs since it is not possible to immediately eliminate all design permutations.

Due to the vast size of the design space, evaluating the lifetime of all designs is generally not possible. The current state-of-the-art optimization techniques use greedy algorithms [6], ant colony optimization [1] and simulated annealing [2]. These algorithms evaluate

a subset of the design-space and strategically select the next set of designs to evaluate. Through this iterative process, the algorithm would converge to the optimal in the ideal case.

In the typical setup, Monte-Carlo Simulation (MCS) is used to estimate lifetime of each of the designs; the designs can then be sorted and the relative lifetime rankings used to indicate the quality. In some optimization algorithms, the exact lifetime does not matter as much as simply identifying a top set of designs for the next iteration; one such example is evolutionary algorithms. Drawing on this observation, the problem lends itself to be formulated as a Multi-Armed Bandit (MAB) problem.

The name Multi-Armed Bandit originates from the casino where a player is playing the slot machine (also called a bandit). The goal is to minimize their regret/loss when pulling the arm of the bandit. The player is allowed to choose from multiple bandits so the problem is a trade-off between exploring other machines, or exploiting the current knowledge of the machines' payouts. An example application of MAB algorithms is in web advertising when the revenue model is pay-per-click. There, the advertising agency wishes to select the most relevant set of ads to display.

Lifetime-evaluation of designs parallels the advertisement selection problem in that the top set out of a larger pool needs to be identified. For the slack-allocation problem, the MAB algorithms will focus the sampling time on areas where it is difficult to distinguish the categorization of the design. In general, MAB algorithms have been shown to offer better performance than uniform sampling in identifying the top arms on Bernoulli distributions [7], [8]. This work applies MAB to the slack allocation problem to attempt to realize sample savings.

1.2 Outline

In this work MAB algorithms are surveyed for the suitability in applications of lifetime-aware design space exploration. MAB algorithms are applied to the slack allocation problem in MPSoCs design. MAB are also used in conjunction with genetic algorithms. Chapter 2 will cover related work. Chapter 3 will discuss the methodology for modeling lifetime. Chapter 4 will discuss the MAB algorithms as well as the associated results. Chapter 5 will contain the setup of the genetic algorithm as well as the results of a direct application of MAB algorithms; finally the conclusion will be in Chapter 6.

Chapter 2

Related Work

2.1 Lifetime-Aware design in MPSoCs

The failure mechanisms of electronic devices have been understood from a materials engineering perspective and a fairly comprehensive list of models have been compiled by the Joint Electronic Device Engineering Council (JEDEC)[9]. Using accelerated failure models, the number of failures in time (FIT) represented by λ can be determined. The time to failure (TF) is calculated from the reciprocal i.e $1/\lambda$. Since the λ represents a random failure rate, this can be easily applied to an exponential failure model.

The earliest application of the exponential failure models into lifetime-aware design was by [10]. Based on failure data collected at IBM, a tool, RAMPS, was developed for modeling system lifetime with 4 underlying failure mechanisms: electromigration, time-dependent dielectric breakdown, stress migration and thermal cycling. Due to the properties of the exponential function, the minimum failure time of the 4 mechanism has a closed form solution which could be used to predict the overall component lifetime. Although exponential models do not provide a good estimate for wear-out failures since they are memoryless,

they provided the groundwork for modeling lifetime in lifetime-aware designs.

The work of [10] was extended in [5] to use log-normal distributions instead. An additional failure mechanism, negative-bias temperature instability (NBTI), was also added. This lifetime model was used in structural duplication (SD) whereby duplicated resources are power gated until they are necessary for surviving failure. The notion of graceful performance degradation (GPD) is also introduced whereby the system continues to operate after failure but at a lower performance. Overall the results show that SD can improve lifetime by 3.17 time but at 2.25 times the area cost. Using GPD gives 1.42 times lifetime improvement but operates on average 5% slower and does not increase the cost. This work shows the validity of using the slack allocation approach; provided the computation tasks deadlines are met, extended lifetime is achievable.

In a recent work by [11], a software method of monitoring the wear of an embedded processor is proposed. The software runs on the operating system and periodically, the system will test a variety of system frequencies and voltages and observe the number of system crashes. This method works best for the NBTI failure mechanism since it manifests as a threshold voltage increase which will affect the system frequency and voltage. A watchdog timer is used to reset the system when a crash is detected and record the failure point. This process was tested on the Intel Atom over 160 days and the failure distributions were obtained for the 3 stages of the device lifetime: early, constant, wear-out failures. This work demonstrates a possible method for obtaining failure distributions as well as for monitoring wear.

Since temperature is a major factor in the failure mechanisms listed by JEDEC [9], the effects of workload also need to be accounted for. If a component failure is survivable, the workload of the failed component will be distributed to remaining working components. The model from [12] accounts for recomputing the failure distribution at the new tempera-

ture (associated with the new workload) adjusted for the accumulated wear for log-normal distributions.

For certain failure-mechanisms, Weibull distributions are a better model than log-normal distributions. In the work of [13], the lifetime is modeled using a mix of log-normal and Weibull distributions for each of the independent components. The overall system failure distribution is found through Monte-Carlo simulation to solve for the aggregate minimum failure time. The main advantage of Weibull distributions is that they provide more accuracy when the number of system components is large whereas log-normal distributions are more accurate when there are fewer system components. By weighing each of the failure distributions models with respect to the number of components, their model obtains an even higher accuracy. In general, in a system with 500 components, the error for both log-normal and Weibull distributions cross over at 10% error when identifying the time where 1% of all devices fail.

Temperature is an important factor affecting lifetime; but the effects of the interrelation of switches and components must also be taken into account. In a temperature minimization simulation for mapping tasks to components, a fluctuation in temperature of 1% led to a lifetime variation of 101.7% [1]. [1] use ant-colony optimization to find the optimal task mapping while modeling system lifetime rather than relying solely on temperature based analysis. When compared to the temperature-aware simulated annealing approach, lifetime-aware task mapping were 17.9% from the optimal and outperformed simulated annealing by 32.3%.

Heuristic approaches have also been proposed for the problem of mapping tasks to processors [14][4]. Using a model which weights power, capacity, temperature, time to failure and wear [14] shows that lifetime aware modeling can also yield 14.6% improvements on average over temperature based heuristics. [4] uses a heuristic for mapping triplicated

tasks onto triple modular redundant systems. By spreading out the tasks mappings and allowing the system to operate in a degraded state as components fail, lifetime can be extended by additional 37%.

A greedy algorithm was proposed by [6] to address the slack allocation problem. Drawing on the observation that the amount of slack to survive failure is fixed to failed component size, there is no benefit of too little or too much slack; this is defined as the critical quantity. By greedily searching designs with slack allocated according to the critical quantities, it was possible to find designs near the cost-lifetime Pareto-optimal front by only evaluating 1.4% of the design space.

Evolutionary algorithms are also a common approach to solving problems with large design spaces. In [15], task mapping is combined with dynamic voltage and frequency scaling (DVFS) to improve reliability in MPSoCs. In order to improve resilience to transient faults, tasks can be duplicated but will lead to higher computation demands decreasing lifetime. To find a good balance between transient fault resilience and device lifetime, non-dominated sorting genetic algorithm II (NSGA-II) is applied to find the set of Pareto-optimal solutions. For simplicity, an exponential model for evaluating lifetime is used, this avoids the need to use MCS for lifetime evaluation.

The approaches by [1], [6], [14], have relied on estimating lifetime by using Monte-Carlo Simulation in order to infer quality of one design choice over another. Since the designers do not care about the exact lifetime but rather a classification of the good from the bad; it is possible to employ the MAB algorithms.

2.2 Multi-Armed Bandits

The original Multi-Armed Bandit (MAB) problem arose from the question of how a player should proceed to minimize their losses when playing the slot machine (bandit) with applications to clinical trials [16], [17], [18]. The best that a player can do is to always be playing the optimal arm, as such the any sub-optimal action leads to an accumulation of regret. Assuming that the underlying reward distributions do not change over time and that the player must play indefinitely, it was shown by [8] that the best arm can be selected asymptotically more often than the inferior arm for select normal, Bernoulli, Poisson and exponential distributions.

One algorithm which was proposed that can achieve the logarithmic bound of [8] for Bernoulli distributions is the UCB1 by [19]. Derived from Hoeffding's inequality, UCB1 is a policy which dictates which arm should be sampled next. A variant of the UBC1 algorithm, UCB2, samples the selected arm multiple times instead of just once in a epoch/phase based approach. UCB2 performs better than UCB1 by a constant factor [19].

Another phase based approach was proposed by [7] in the successive-rejects (SR) algorithm. This algorithm relaxes the assumption of the time horizon and instead operates in a fixed time budget. The first step is to allocate the time budget into separate phases. At each phase, the associated sample budget is expended into all the remaining arms (in this case, designs). The worst design at the end of the phase is rejected. This algorithm iterates until there is only a single design remaining.

For lifetime-aware design, identifying the single best design is not as useful as identifying a top set of m designs. The Successive-Accept-Reject (SAR) algorithm in [20] addresses this shortcoming. The original SR algorithm is modified so that at the end of each phase, the design farthest from the mth design is either accepted or rejected. Evaluation on the

accepted or rejected design stops, and the sampling continues on the remaining designs until m designs have been found.

In the work by [21], the Gap based exploration (GapE) algorithm was created which extends the UCB1 to work for the problem of identifying the top arm out of separate population of bandits. The work of [22] parallels [21] but rather the problem is stated as finding the top set of arms. The work of [22] also creates a unified algorithm which can operates in either a fixed budget constraint where the number of pulls is predetermined or stop once a desired confidence level has been reached.

Although [22] is more difficult to implement compared to the SAR algorithm and requires tuning parameters, it has been shown to have better performance on select distributions [20]. It is however possible to use adaptive parameters to avoid tuning altogether [22].

2.3 Contribution

This work is, to the author's knowledge, the first application of MAB algorithm to the lifetime-aware slack allocation problem. The contribution is as follows:

- MAB algorithms are adapted for use in lifetime evaluation and the benefits are assessed over uniform sampling for lifetime evaluation. Overall MAB lead to a 1.45x-5.26x speedup in by reducing the amount of samples needed to reach the same level of confidence selection
- MAB is compared against uniform sampling when used as a selection algorithm in an evolutionary algorithms. In this setup, MAB is shown to be no more effective of a selection algorithm than MCS.

Chapter 3

Methodology – Lifetime Modeling

The simulator which is used to evaluate lifetime is based on [6] with modifications to facilitate MAB lifetime evaluation. This chapter will present an overview of the inner workings of the simulator.

3.1 Failure Mechanisms Models

In the interest of determining the lifetime of MPSoCs it is important to understand the types of faults which can be prevented. In general, there are 2 types of errors commonly found in processors: soft errors, hard errors [23].

Soft Errors Soft errors are transient errors which can occur when radiation strikes the processor and causes a temporary upset in the form of bit flips. When the data is corrupted, the processor can continue to function if there are ecovery mechanisms are in place. If no recovery mechanism is available, the circuit will continue to function normally after power cycling.

Hard Errors Hard errors are errors which are part of the material properties of the processor and are permanent. There are 2 types of hard errors, extrinsic failures and intrinsic failures. Extrinsic failures are typically due to manufacturing defects and are decreasing over time. Intrinsic failures correspond to the wear-out of a processor and are increasing over time.

Since the lifetime of a component will primary depend on the wear out characteristics, only intrinsic errors are modeled. A comprehensive list of common failure mechanisms for electronic devices is available from [9], but only a selection of these are used in the simulator. They are discussed in more detail in the following sections and are eletromigration, time dependent dielectric breakdown, thermal cycling and stress migration. This choice maintains consistency with prior work from IBM where fitting data was available [10].

Electromigration (EM)

Electromigration (EM) is a mechanism which describes the creep of material in the form of voids and hillocks due to movement of metal ions. The movement is caused my momentum transfer from electrons to the positive ions under the presence of a current. A popular model for measuring the time to failure of EM is Black's equation [9].

$$TF_{EM} = \mathbf{E}[X_{EM}] = \frac{A_{EM}}{(J - J_{crit})^n} \exp\left(\frac{E_a}{kT}\right) \tag{3.1}$$

Where

- TF_{EM} is the expected time to failure of the mechanism with random variable X_{EM}
- A_{EM} is a constant based on the cross-sectional area of the interconnect
- J is the current density [A/cm^2]

- J_{crit} is the critical current density [A/cm^2]
- E_a is the activation energy (typical values 0.5-0.6 eV Al+Si; 0.7-0.9 eV Al+Cu)
- k is the Boltzmann constant (8.617×10^{-5} eVK^{-1})
- T is the temperature [K].
- n is a scaling factor (typically set to 2)

Time dependent dielectric breakdown (TDDB)

Time dependent dielectric breakdown describes the breakdown of a dielectric material between two conductors. This is generally seen in the gate oxide where microscopic holes are formed [9]. The simulator uses an empirical model to estimate the time to failure based on the works of [10] and the model is given in equation 3.2

$$TF_{TDDB} = \mathbf{E}[X_{TDDB}] = A_{TDDB}\left(\frac{1}{V}\right)^{(a-bT)} \exp\left(\frac{X + \frac{Y}{T} + ZT}{kT}\right) \tag{3.2}$$

Where TF_{TDDB} is the expected time to failure of the mechanism with random variable X_{TDDB}. The equation has the fitting parameters a, b, X, Y, Z where $a = 78,\ b = -0.081,$ $X = 0.769eV,\ Y = -66.8eV,\ Z = -8.37 \times 10^{-4}eV/K$, V is the voltage, T is the temperature, and A_{TDDB} is a constant.

Thermal cycling (TC)

Thermal cycling is a failure mechanism due to fatigue that occurs due to cycling of temperature. The number of cycles until failure is given in (3.3) [9].

$$N = \frac{C_0}{(\Delta T - \Delta T_0)^q} \tag{3.3}$$

Where

- C_0 a material dependent constant
- ΔT is the entire temperature cycle-range for the device
- ΔT_0 is the portion of the temperature range in the elastic region
- N is the number of cycles to failure
- q is the Coffin-Manson exponent, an empirically derived constant (typically 3-5 for Al,Au)

A simplification is used in the simulator based on the model from [10], equation 3.3 is simplified to equation 3.4 where TF_{TC} is the expected time to failure of the mechanism with random variable X_{TC}, $q = 2.35$ and A_{TC} is a scaling constant

$$TF_{TC} = \mathbf{E}[X_{TC}] = \frac{A_{TC}}{\Delta T^q} \tag{3.4}$$

Stress migration (SM)

Stress migration describes the movement of metal ions due to mechanical stresses; the stresses are typically thermally induced [9].

$$TF_{SM} = \mathbf{E}[X_{SM}] = A_{SM} \frac{1}{|T_0 - T|^n} \exp\left(\frac{E_a}{kT}\right) \tag{3.5}$$

Where

- TF_{SM} is the expected time to failure of the mechanism with random variable X_{SM}
- A_{SM} is a constant
- T_0 is stress free temperature for the metal (metal decomposition temperature)

- n is a constant (typically 2-3)
- E_a is the activation energy (typically 0.5-0.6 eV)
- k is the Boltzmann constant (8.617×10^{-5} eVK^{-1})
- T is the temperature [K].

3.2 Failure Model for Component lifetime

The failure of devices throughout lifetime generally falls into 3 phases:

- Early Failure (decreasing failure rates)
- Random failures (constant failure rates)
- Wear-out failures (increasing failure rates)

When the number of failures for a system is plotted against time, the curve that is traced out resmbles a bathtub and is typically referred to as the *bathtub curve*. Early failures represents a region where the failures are decreasing with time, typically associated with manufacturing defects. Wear-out failures are increasing over time and represent the failures due to aging. Finally, devices will fail randomly which are neither classified as early or wear-out failures. Each phase of failure can be modeled using a probability distribution; some of the most common models are exponential, lognormal and Weibull distributions.

The distributions will have different parameters, but the general assumption is that the TF obtained from the previous section will be used as the mean of the distribution (i.e. the expectation of the random variable $X_\bullet$ for each failure mechanism). In section 3.1 each mechanism had an estimated time to failure scaled by a constant $A_\bullet$; the subscript is used to denote the respective failure mechanisms. $A_\bullet$ is a process dependent constant and since this data is difficult to come by, some assumptions were made. The value of $A_\bullet$

was calculated assuming a failure time of 30 years when characterized at a temperature of $345K$; this is consistent with prior work which uses these models for lifetime [10]. While the choice of 30 years is arbitrary, the assumption is that at 30 years time, circuits will have entered the wear out region and random failures will no longer be the dominant failure mechanism.

The time to failure of a component will be dictated by the minimum of the random variable, $X_\bullet$, with a corresponding failure distribution per mechanism (3.6). In cases where no closed form solution exists, Monte-Carlo Simulation (MCS) is used to estimate the distribution of random variable for the component lifetime, Y_c.

$$\mathrm{Y}_c = \min\left(X_{EM}, X_{TDDB}, X_{TC}, X_{SM}\right) \tag{3.6}$$

3.2.1 Exponential Model

The exponential model is the simplest failure model and has the *pdf* given by (3.7). It is specified with one parameter λ which represents the number of failures in time (FIT) over 10^9 hours [9]. The parameter λ can be obtained by taking the reciprocal of the TF i.e. $\lambda = 1/\mathrm{TF}$.

$$f(t;\lambda) = \lambda e^{-\lambda t},\ t \geq 0 \tag{3.7}$$

Since the overall component lifetime is dictated by the minimum failure time of each of the mechanisms, this model has a closed form solution. In particular, the minimum of exponential random variables with rates λ_1 and λ_2 is an exponential random variable with rate $\lambda_1+\lambda_2$. Therefore, the component failure time, TF_c, is an exponential random variable with rate $\lambda_{EM} + \lambda_{TDDB} + \lambda_{TC} + \lambda_{SM}$.

The exponential model is good for modeling the random failures but since it has the

memoryless property, it is not suitable for use for when modeling wear out failures.

3.2.2 Log-normal

The log-normal distribution has the *pdf* in (3.8) and has 2 parameters μ and σ.

$$f(t;\mu,\sigma) = \frac{1}{\sigma t\sqrt{2\pi}} \exp\left(- \frac{(\ln t - \mu)^2}{2\sigma^2} \right),\ t > 0 \tag{3.8}$$

In addition, log-normal distributions have the following properties

- The mean is $e^{\mu+\sigma^2/2}$
- The median is e^{μ}
- The variance is $(e^{\sigma^2} - 1)e^{2\mu+\sigma^2}$

For lifetime modeling, [9] recommends the parameter μ to be set to the median failure time. The shaping parameter σ, typically has a value < 1 when modeling wear-out failures [24]. [25] recommends setting the value to $\sigma = ln(t_{50}) - ln(t_{16})$ where t_{50} is the median or 50th percentile and t_{16} is the 16th percentile. For lifetime simulation by [5] a value $\sigma = 0.5$ is chosen.

3.2.3 Weibull

Weibull is a probability distribution best used to model a scenario of the weakest link, i.e failure of the weakest component results in failure of the entire system. The *pdf* for Weibul distributions is given in (3.9) and has parameters α and β.

$$f(t;\alpha,\beta) = \frac{\beta}{\alpha}\left(\frac{t}{\alpha}\right)^{\beta-1} e^{-(t/\alpha)^{\beta}},\ t \geq 0 \tag{3.9}$$

The parameter β is a shaping parameter which determines if the rate of failure is increasing ($\beta < 1$), constant ($\beta = 1$), or decreasing ($\beta > 1$) over time. For lifetime modeling $\beta = 5$ is appropriate [13]. The parameter α is a scale parameter, and for lifetime distributions is the characteristic time-to failure; it should be set as the point where 63% of devices fail [25].

3.2.4 Adjusting distributions post-failure

In the current setup, the lifetime distributions means are tuned to the lifetime estimate from each of the failure mechanisms at a characterization temperature. This model can be used to find the time to failure but does not assume failure is survivable.

When the system survives failure, the workload of the failed component is transferred to the remaining working components. This increase in activity will increase power and temperature and hence accelerate failure further. The remaining components at the time of failure will have already accumulated an amount of wear and hence the distribution will need to be adjusted to reflect this.

Given that the probability distribution of a failure mechanism at temperature T_1 is $f_1(t)$ at time t_1 and that the new operating temperature is T_2 with the distribution $f_2(t)$, then (3.10) should be satisfied [12]. In short, the overall probability density function should integrate to 1 including the discontinuities in temperature.

$$\int_0^{t_1} f_1(t)dt + \int_{t_1}^{\infty} f_2(t)dt = 1 \tag{3.10}$$

The equation can be generalized to the scenario where there are multiple failures or operating temperatures.

$$\int_{t_{s0}}^{t_{s1}} f_0(t)dt + \int_{t_{s1}}^{ts2} f_1(t)dt + ... + \int_{t_{N-1}}^{\infty} f_{N-1}(t)dt = 1 \tag{3.11}$$

3.2.5 Choosing A Model

Since the primary concern is modeling the wear-out lifetime, the exponential distribution is not a good choice; log-normal distributions are used in the simulator to be consistent with [10]. It is worth noting that a combination of distributions can be used; in [13], EM is modeled with log normal distributions and Weibull distributions for the others mechanisms. Overall, the process of calculating the component time to failure TF_c remains the same as it is done through MCS.

3.3 System Lifetime

In section 3.2, an overview of the distributions fitted to the failure mechanisms was presented. This section will discuss the details for estimating system lifetime of an MPSoC from a component level understanding.

Every problem begins with the definition of the workload and the system architecture. The workload describes the application which is to be run and the system architecture describes the inter-connectivity of the components within the MPSoC. In the problem of slack allocation, the choice of component is flexible and the inter-connectivity between components is fixed. As slack is allocated, the physical die area will increase. With each new slack allocation, a new physical layout is required, these new floor-plans are generated automatically using Parquet [26].

The applications which are run on the system are broken down into smaller tasks. These are mapped into the components. As was shown by [1], [14], [4], the choice of the task

mapping can affect the lifetime but choosing an optimal task mapping is beyond the scope of this work; an arbitrary task mapping consistent with [6] was used.

From the characteristics of the components, the power consumption of the component can be determined. This information, in conjunction with the floorplan, can be used in HotSpot, a thermal simulator, to determine the component temperatures [27]. The temperature is used to estimate the failure mechanism lifetimes which parameterize the failure distributions. By finding the failure time of the mechanisms and the corresponding component failure, the system lifetime can be estimated.

This section will first outline how the system lifetime is calculated; followed by an overview of the workloads and system architectures which are simulated.

3.3.1 System Lifetime

Once a system has been initialized (valid architecture, floor planned, tasks mapped, power estimated), the system lifetime can be estimated using MCS. The failure mechanism distributions are initialized to their operating temperature and a random sample for the time to failure (TF) is taken; the minimum of the failure times is the components failure time (3.12).

$$\mathrm{TF}_C = \min\{\mathrm{TF}_{TDDB}, \mathrm{TF}_{SM}, \mathrm{TF}_{EM}, \mathrm{TF}_{TC}\} \tag{3.12}$$

The lifetime of the overall system is determined by the aggregate failure of the components. If there were no redundant resources in the system, then failure of any component would result in the failure of the system and the system lifetime would be the minimum of the component failure times (3.13).

$$\mathrm{TF}_{system} = \min\{\mathrm{TF}_{C1}, \mathrm{TF}_{C2}, ..., \mathrm{TF}_{Cn}\} \tag{3.13}$$

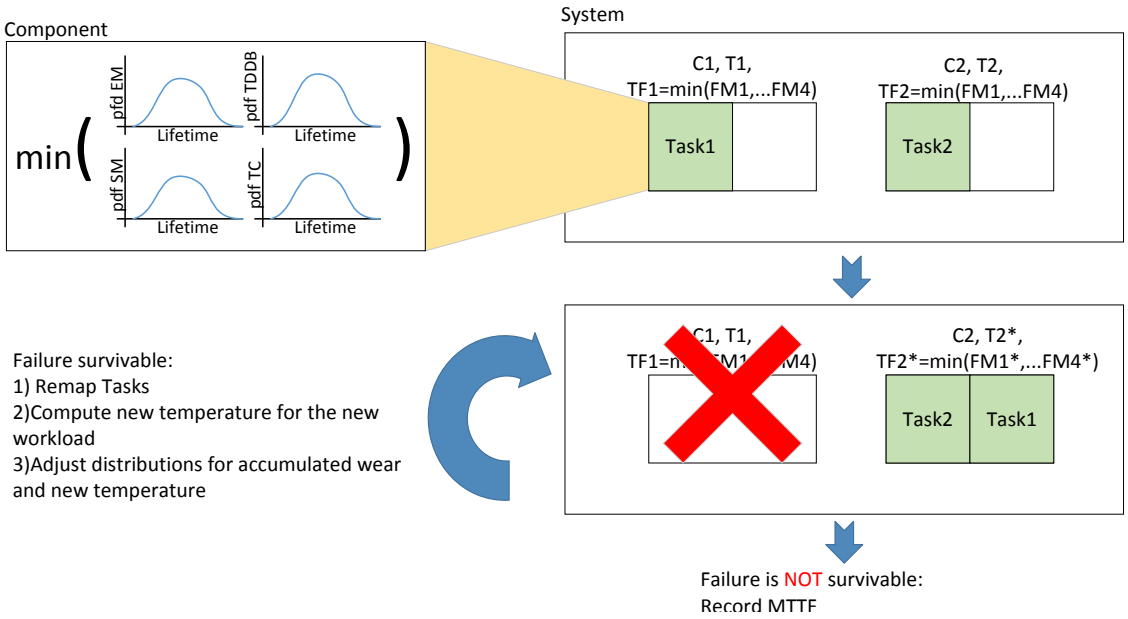


Fig. 3.1 The system lifetime is estimated using MCS. Each component lifetime is estimate through the time to failure of 4 mechanism: EM, TDDB, SM, TC. The MTTF is determined when the system can no longer survive failure and is the average of many samples.

However, if the failure is survivable, the tasks are remapped to working components. If multiple survivable mappings exists, the task will be mapped to the component with the fewest hops in communication from the failed component to minimize traffic; this assumes that the initial mapping provided to the system is sensible.

Using the new workload, a new power estimate is generated. The lifetime distributions are adjusted to reflect the new operating temperature and accumulated wear as outline in section 3.2.4. These steps are repeated until failures are no longer survivable. This process is repeated many times for a given system and the mean of the TF_{system} is the mean time to failure (MTTF). The necessity of MCS is apparent since it is difficult to obtain a closed

form mathematical model for the system lifetime. The entire process is roughly outlined in figure 3.1.

3.3.2 Workload

The workloads which are simulated are multi-window display (MWD) and a MPEG decoder. MWD is an application which composites multiple video sequences into a single display [28]. MPEG is an implementation for the MPEG4 encoding standard [29]. The task graphs for both of these workloads are shown in Figures 3.2-3.3 as reproduced by [30].

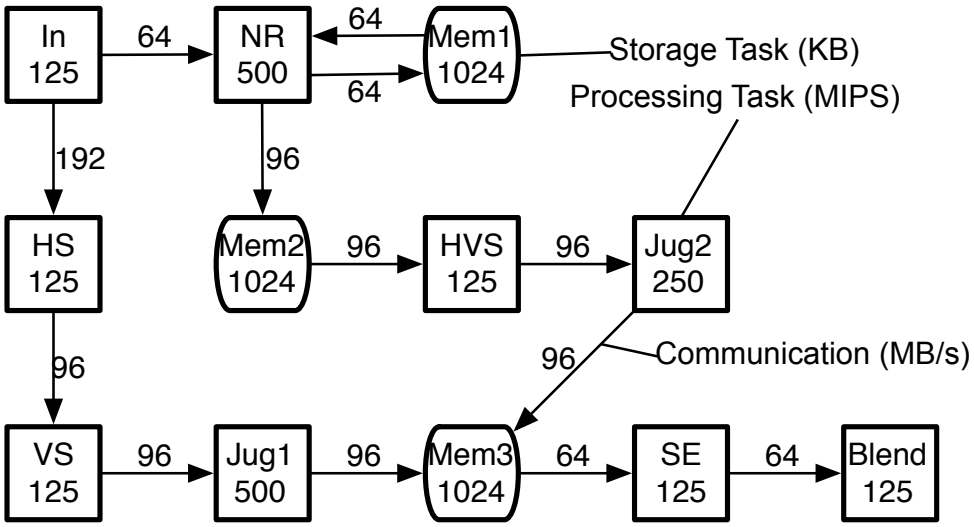


Fig. 3.2 MWD task graph [30]

3.3.3 Architecture

Each of the workloads are run on 2 systems architectures consistent with [30]. The tasks of the task graph are arbitrarily mapped to the components while minimizing the number of hops in communication. The MWD3S and MPEG4S consist of the minimum number

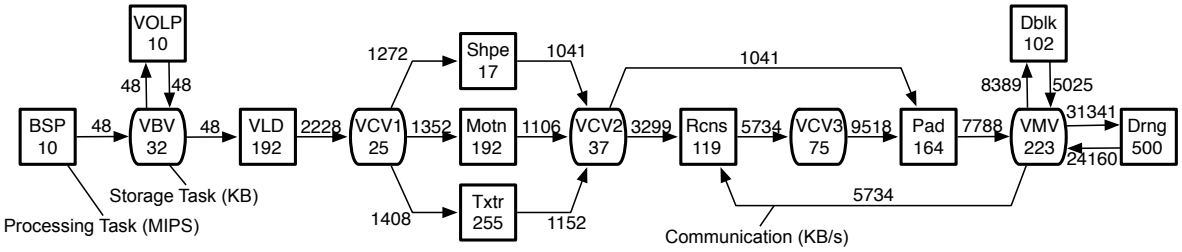


Fig. 3.3 MPEG task graph [30]

of switches to connect all the components together; in this case a switch failure is not survivable. In MWD4S and MPEG5S the switches are configured in a ring architecture so that one switch failure may be survivable.

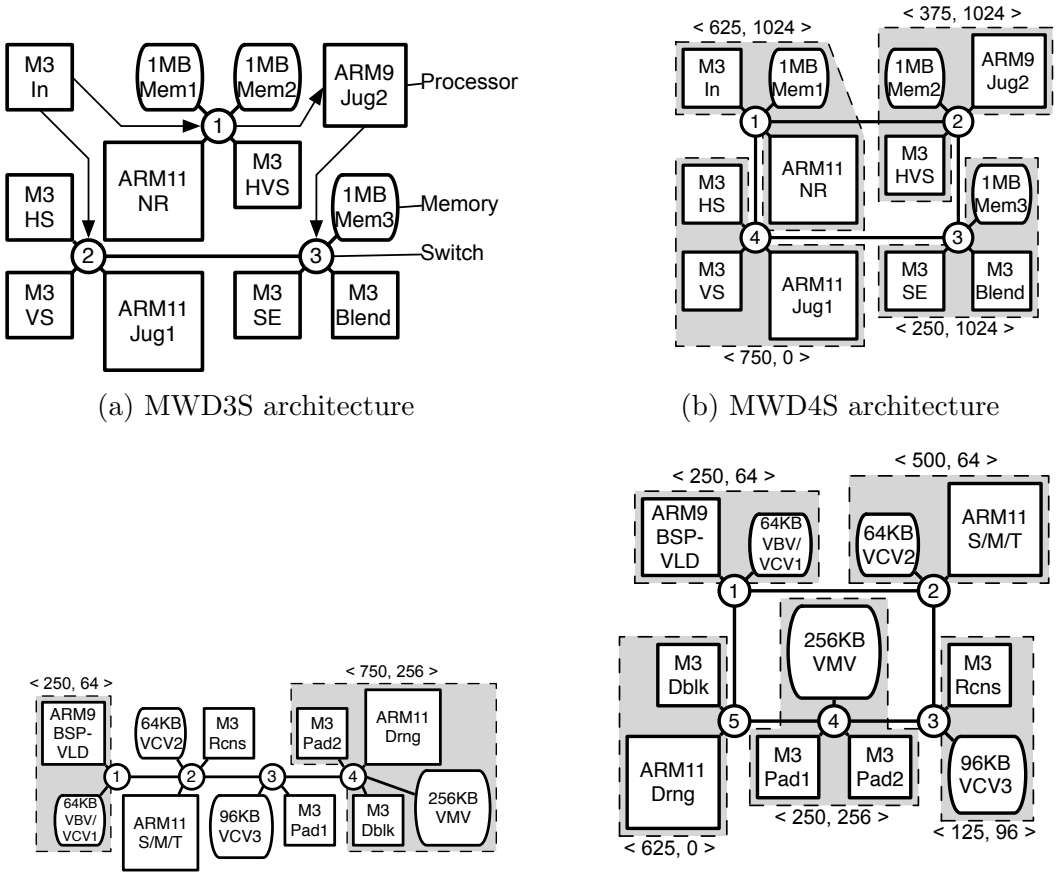
3.3.4 Components

In the MPSoC lifetime simulation, there are three types of components which are modeled: switches, memories and processors. For each type of component, there are multiple upgrade options which can be found from the component library. A list of the available components can be found in table 3.1.

Table 3.1 The 3 classes of components which are modeled in the simulator as well as the respective upgrade configurations

Component Class	Configurations
Processors (3)	ArmM3 (125 MIPS), Arm9 (250 MIPS), Arm11 (500 MIPS)
Memories (9)	64kB, 96kB, 128kB, 192kB, 256kB, 384kB, 512kB, 1MB, 2MB
Switches (2)	4x4, 5x5

The static power, dynamic power and area of each of the components can be estimated. For the processors, this information is available from the data sheets; for memories they are obtained using CACTI 5.3, for switches ORION 1.0. A detailed summary of the actual values is available in [30].



(a) MWD3S architecture

(b) MWD4S architecture

(c) MPEG4S architecture

(d) MPEG5S architecture

Fig. 3.4 Figures show the task mapping and architectures for the various benchmarks [30]

The number of slack allocations which exist in the benchmarks is dependent on the number of upgrade options for the processors and memories associated with a baseline design. The switches are not changed for the slack allocation problem since they are part of the system network architecture. The baseline architectures are as follows:

- MWD: 6×Arm M3, 1×Arm9, 2×Arm11, 3×1MB mem
- MPEG: 4×Arm M3, 1×Arm9, 2×Arm11, 2× 64kB, 1×96kB, 1×256kB

For MWD, the number of slack allocations are: $3^6 \times 2^1 \times 1^2 \times 2^3 = 11664$. For MPEG the maximum memory size permitted is restricted to 512kB and the number of slack allocations are: $3^4 \times 2^1 \times 1^2 \times 7^2 \times 6^1 \times 3^1 = 142884$. Table 3.2 summarizes the benchmarks.

Table 3.2 Simulated workloads and NoC architectures. The number of designs (slack-allocations) for each benchmark is also listed

Benchmark	NoC Architecture	Processors	Memories	Designs
MWD3S	3-switch 1x3	9	3	11664
MWD4S	4-switch ring	9	3	11664
MPEG4S	4-switch 1x4	7	4	142884
MPEG5S	5-switch ring	7	4	142884

This chapter has presented an overview of the simulator mechanics for lifetime estimation. The benchmarks used on the various design architecture as well the number of design permutations were discussed. In the next chapter, MAB strategies to find designs with the highest lifetime in the vast design space will be explored.

Chapter 4

Multi-Armed Bandit

In the previous chapter, the mechanics of the lifetime simulator was presented along the scope of the slack allocation design space. The problem starts with a baseline design which meets the basic computation requirements of an application; to improve lifetime, the components can swapped with ones of higher capacity. Since there exists many component types the number of permutations is large analyzing all the design permutations is time consuming. The naive approach would be to evaluate the lifetime of all design using MCS. Given that a designer will work with a smaller group of designs where the goal is to narrow it down to the best designs, MABs are a more efficient approach. This chapter outlines the setup of Multi-Armed Bandits (MAB) for the slack allocation problem as well as the results.

In this work, we will assume that there is a fixed amount of time (number of samples) that is allowed for the top designs to be identified. The designer is interested in identifying a set of designs rather than the single best design; this allows the designer to factor in other practical issues that might make a design less favorable than another (such as quality of the floor plan). As such, this establishes some requirements for the MAB algorithms which

are used.

1. The cost of each sample remains constant over time
2. The underlying lifetime distributions do not change when sampled
3. There is a fixed time/sampling budget of n samples
4. The top set of m designs must be identified out of a set of D designs

Naturally, items 1 & 2 are consistent with classical bandit algorithm definitions such as the UCB1 [19]. In this setup, the goal is to minimize the regret when sampling over an infinite time period. To do this, the single optimal arm should be sampled more often than others (Figure 4.1). This is however not as useful for the slack allocation problem since there is a fixed amount of time. It is also desirable for a set of designs to be discovered.

For MAB to be used for the slack allocation problem, algorithms which satisfy items 3 & 4. The literature was surveyed and 2 suitable MAB algorithms were identified. In the following sections the MCS (uniform sampling) algorithm as well the 2 suitable MAB algorithms are presented for this problem.

4.1 Uniform sampling

The simplest approach that does not involve any strategic sampling is to distribute the samples uniformly across all designs. This serves as the baseline approach that is to be improved upon and is shown in algorithm 1.

As alluded to earlier, this approach is sub-optimal if the problem is to classify the bad designs from the good designs. The reason is that the worst designs are sampled as long as the good designs. A better optimal choice would be to focus the samples at the decision boundary of the good and bad designs.

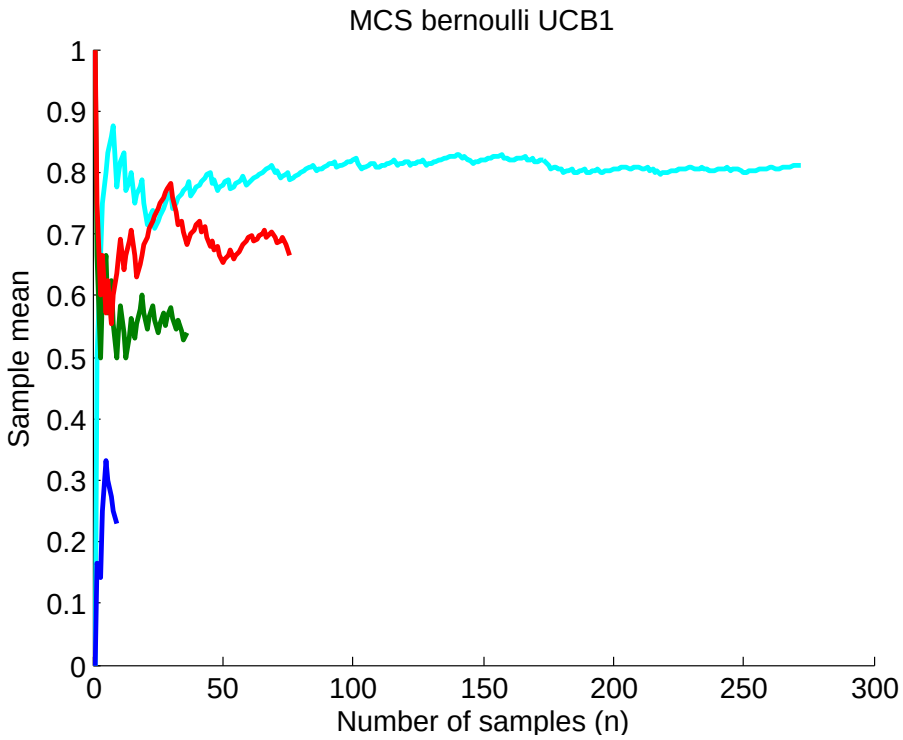


Fig. 4.1 In UCB1, the samples are primarily spent on the best arm. Four arms of probabilities [0.8, 0.7, 0.5, 0.3], the UCB1 policy samples each of the arms but spends most of the time exploiting the best arm.

4.2 Sucessive Accept Reject (SAR)

A suitable algorithm for this application is the Successive Accept Reject (SAR) shown as algorithm 2 and is one formulation of non-uniform sampling [20].

The SAR algorithm starts with all designs being marked as active. The sample budget is divided into $D-1$ separate phases. The budget allocated at a particular phase is spent evaluating all active designs at which point the algorithm makes a decision to deactivate the worst or the best design permanently; this decision is based on whichever has the farthest mean lifetime from the mth design. This process is continued for all phases. The intuition is that the best and the worst stand out and they are easy to identify. If a classification

error were to occur it would most likely be the $m \pm 1$th design.

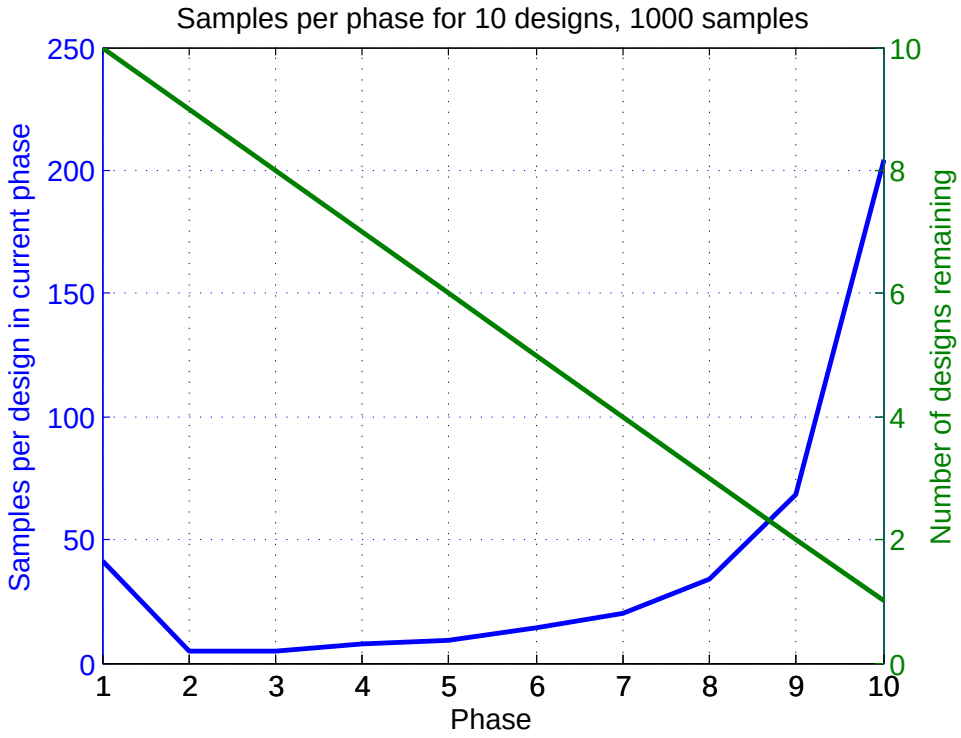


Fig. 4.2 Plot shows the samples allocated per design at each phase for a group of 10 designs where the total sample budget is 1000 samples

The allocated budget per-phase is calculate as $n_k - n_{k-1}$ where k is the current phase. This quantity grows exponentially with the exception of the first phase (figure 4.2). Using this budgeting strategy, [20] has shown that the regret is logarithmic bounded.

One of the drawback of the SAR algorithm is that a design is permanently removed from evaluation at each phase. If a mis-identification occurs, the design cannot be classified correctly again. An algorithm without this shortcoming is the GapE algorithm provided by [21].

4.3 GapE

The GapE algorithm was originally formulated for the purpose of clinical trials. Given that there are multiple sub-populations each with multiple treatment options, what is the best strategy for finding the best treatment for each sub-population [21]. This problem shares parallels with the problem of identifying the top m arms and was extended to capture this scenario in [22]. Although there are multiple versions of the GapE algorithm, this work uses the most extended version of the algorithm which provides the versatility of self-tuning parameters and also captures the variance of each of the sampled arms.

The principle behind the GapE algorithm is the B-index, a function $B_i(t)$ which quantifies which design should be chosen next. The next design to evaluate should maximize the B-index (4.1).

$$I = \text{argmax}_i \mathbf{B}(t) \tag{4.1}$$

$$B_i(t) = \underbrace{-\widehat{\Delta}_{mi}(t-1)}_{\text{Empirical gap}} + \underbrace{\sqrt{\frac{2a\widehat{\sigma}_i(t-1)}{T_i(t-1)} + \frac{7ab}{3(T_i(t-1)-1)}}}_{\text{Exploration factor}} \tag{4.2}$$

The B-index typically consists of 2 components: the *empirical gap* which weights the quality of a design and the *exploration factor* which weights the time spent evaluating the design i.e the knowledge of the other designs. The B-index used in the variance based GapE is shown in (4.2) where:

- $a = \frac{8}{9}\frac{n-2D}{H^{\sigma}}$
- b is an upper-bound on the lifetime distributions over the range [0,b]
- t is the current sample
- $T_i(t)$ is the number of samples spent on design i at sample t
- $\widehat{\sigma}_i(t)$ is the variance of the lifetime of design i at sample t

- $\widehat{\Delta}_{mi}(t) = |\widehat{\mu}_i - \widehat{\mu}_m|$ is the distance of the i^{th} mean from the m^{th} boundary at sample t

Consider the effects of the *empirical gap* term: if a design is close the the mth boundary, the $\widehat{\Delta}_{mi}$ estimate will be closer to 0; if a design is farther from the mth boundary, it will be much larger. When taking the argmax, designs with smaller $\widehat{\Delta}_{mi}$ will subtract less from the overall expression. This implies that designs closer to the boundary are more likely to be selected. This is inline with the intuition that the challenging part of the problem is deciding if the mediocre designs belong in the good set or the bad set.

The purpose of the *exploration factor* is to minimize the uncertainty on designs that have not received much sample time. There are 2 key parameters, the variance $\widehat{\sigma}_i(t)$ and the number of samples spent $T_i(t)$ on design i at time t. When the variance is high, B_i is larger than when the design has a lower variance implying it is more likely to be sampled. When a design has received fewer samples $T_i(t)$, the smaller value in the denominator makes B_i larger than if $T_i(t)$ were larger. The combined of effects of these two parameters affects the outcome of which design will be sampled next.

The parameter a is further defined by H^σ which reflects the hardness of the distribution and is formally given by equation 4.3.

$$H^\sigma = \sum_{k=1}^{D} \frac{(\sigma_{mk} + \sqrt{\sigma_{mk}^2 + (16/3)b\Delta_{mk}})^2}{\Delta_{mk}^2} \tag{4.3}$$

The value of this parameter usually unknown unless the user has prior knowledge of the problem distribution; it is however possible to use an adaptive estimate. It is shown by [21] that it is sufficient to use the estimate in equation 4.4.

$$\widehat{H}^\sigma = \sum_i \frac{\left(LCB_{\sigma_i}(t) + \sqrt{LCB_{\sigma_i}(t)^2 + b(\frac{16}{3})UCB_{\Delta_i}(t)}\right)^2}{UCB_{\Delta_i}(t)^2} \tag{4.4}$$

where the $UCB_{\Delta_i}(t)$ and $LCB_{\sigma_i}(t)$ are given by equations 4.5, 4.6 respectively.

$$UCB_{\Delta_i}(t) = \widehat{\Delta}_i(t-1) + \sqrt{\frac{1}{2T_i(t-1)}} \tag{4.5}$$

$$LCB_{\sigma_i}(t) = \max\left(0, \widehat{\sigma}_i(t-1) - \sqrt{\frac{2}{T_i(t-1)-1}}\right) \tag{4.6}$$

At every sample of the GapE algorithm (algorithm 3), the $\mathbf{B}(t)$ at the corresponding sample t is calculated. The next design is chosen to maximize $\mathbf{B}(t)$. The mean and variance of the designs are recomputed followed by the gaps of the means $\widehat{\mathbf{\Delta}}_m$. This process iterates until the entire sample budget is expended. The designs can then be sorted and the top set of m designs extracted.

The GapE algorithm naturally runs on an infinite time horizon since the selection process dynamic with each sample, nonetheless in [22], it is shown that in the fixed budget problem, the simple-regret is also bounded. A variant of the algorithm which samples until a fixed confidence is reached is also proposed. Although the GapE algorithm requires more computation between samples compared to either the SAR or MCS, if the sampling time is sufficiently large this effect is negligible.

4.4 Comparison of algorithms

The SAR and GapE share similarities with each other and with traditional bandit algorithms but there are some subtle differences; a summary is given in table 4.1.

Constraints As mentioned earlier in the constraints, the problem must have a fixed cost of sampling and the reward distribution cannot change. This may not be true in all applications (such as clinical trials) but in the application of lifetime simulation, collecting

Table 4.1 A summary of the similarities and difference between the proposed bandit algorithms and traditional bandit algorithms. The UCB1 algorithm is used as the reference for a traditional algorithm [19]

	Traditional (UCB1)	SAR	GapE
Cost of sampling	Fixed	Fixed	Fixed
Static reward distribution	Static	Static	Static
Can find best arm	Yes	Yes	Yes
Can find best set of arms	No	Yes	Yes
Infinite Time Horizon	Yes	No	Yes
Fixed Time Horizon	No	Yes	Yes
Fixed Confidence	No	No	Yes
Minimization goal	Cumulative Regret	$\mathbb{P}$(Mis-identification)	Simple Regret
Sampling strategy	argmax	phase	argmax

samples are constant cost and there is no inflation over time.

Number of arms All of the mentioned bandit algorithms are able to identify the single best arm. Only the SAR and GapE algorithm can identify the top set of arms. The SAR algorithm is only able to do so when the sampling budget is known. The GapE algorithm is more versatile in that, it can work in either case; in an extension of the algorithm, it can also stop when the answer is within a certain confidence threshold.

Minimization Goal The minimization goal for each of the bandit algorithms is different. In the traditional bandit problem, there was a regret associated with the current action as well as all the previous sub-optimal actions; this is summarized by the cumulative regret (4.7).

$$R_n = \sum_{t=1}^{n} \mu^* - \mu_{I(t)} \tag{4.7}$$

The cumulative regret at time n is equal to the sum of the differences between the reward of the optimal arm μ^* and the reward of a selection policy $I(t)$.

In the problem of identifying the top set of arms, the goal is to recommend a set of arms, as such there is no regret for previous actions; the simple regret captures this more effectively (4.8).

$$r_n = \sum_{i=1}^{m} \mu_i^* - \mathbb{E}\mu_{J(i)} \tag{4.8}$$

For each of the top m arms, the regret at sample n is the difference between the optimal mean and the expected mean of a selection policy $J(i)$. If the quality of results are independent of the regret, the algorithm can minimize the probability of mis-identification instead (4.9). This is the goal of the SAR algorithm.

$$e_n = \mathbb{P}[\{J_1, ..., J_m\} \neq \{1, ..., m\}] \tag{4.9}$$

The GapE uses a variant of the simple regret where the regret of a set of chosen arms $\Omega(t)$ at time t is the difference between the m^{th} arm with the optimal mean μ_m^* and the minimum mean from the selected set.

$$r_{\Omega(t)} = \mu_m^* - \min_{i \in \Omega(t)} \mu_i \tag{4.10}$$

The regret for choosing a set $\Omega(t)$ is the regret associated with the worst arm in the set or the maximum regret seen in all the individual arms. The exact quantity of regret may not be useful if the goal is to reach a desired level of accuracy ϵ. In [22], a probability measure is used (4.11). For this algorithm, it is possible to continue sampling until a desired target for δ is reached.

$$\mathbb{P}[r_{\Omega(n)} > \epsilon] \leq \delta \tag{4.11}$$

Sampling Strategy Each algorithm uses a different sampling strategy. In the UCB1 and GapE algorithm, the next arm to sample is chosen by finding the argmax of a UCB-like equation. A decision of the arm to choose is made at each sample. The SAR algorithm breaks the samples into phases and only makes a decision at the end of the phase. The samples spent at each phase will not change if the sample budget, total arms and top arms (m) remains constant. Tables 4.2 & 4.3 show an example of the sample distribution of both the algorithms for a uniformly distributed group of Bernoulli bandits.

Table 4.2 Percentage of time spent evaluating each arm for top 50% of arms. $n = 10,000$

Method \ℙ	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Uniform	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00
SAR	4.58	5.14	6.86	13.71	20.58	20.58	10.29	8.22	5.88	4.12
GapE	2.43	3.45	3.79	8.62	32.04	32.03	7.32	3.98	3.02	2.32

Table 4.3 Percentage of time spent evaluating each arm for top 20% of arms. $n = 10,000$

Method \ℙ	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Uniform	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00
SAR	4.12	4.58	5.14	5.88	6.86	8.22	10.29	20.58	20.58	13.71
GapE	1.40	1.60	1.85	2.08	3.20	3.90	9.29	34.24	34.19	8.25

4.5 Experimental Setup

To gauge the effectiveness of the algorithms, the true lifetime of all slack allocations for each benchmark needs to be estimated. The reference lifetime was obtained by collecting 1 million MCS samples; this is equivalent to the 95% confidence interval being within 0.005 years.

For each of the benchmarks, 100 designs were randomly selected and the top group of designs was identified using the 3 methods: MCS, SAR, GapE. Each permutations of the 100 designs was simulated 100 times. This process was repeated over 100 random design permutations for a total of 10,000 simulated points. The sample budget was swept over a range of values.

4.5.1 Metrics

The metric used to measure the quality of the designs is a combined version of the simple regret and a probability measure that the results are within a certain error (4.12). The simple regret, r_n, is the difference between the sum of the means of the optimal set from the chosen set; ϵ is an error bound parameter; δ is a confidence parameter.

$$\mathbb{P}[r_n > \epsilon] \leq \delta \tag{4.12}$$

This metric is used instead of the simple regret since it is more interesting to know if a mis-identification has occurred rather than the amount regret from choosing a set of designs. This metric assumes that there is a target level of accuracy that is needed for the aggregated lifetimes of the chosen set. If r_n is larger than ϵ, then the set of designs contains enough bad designs that it falls out of the scope of the requirements. The regret $r_{\Omega(t)}$ used in GapE (4.10) is not sufficient to capture the complexity of this problem since it only measure regret for the single worst arm.

The choice for the value of ϵ is dependent on the application. A value of $\epsilon = 1$ was chosen as a conservative estimate since it reflects a 10% accuracy assuming an average device lifetime of 10 years.

4.6 Results and Discussion

The lifetimes of each of the benchmarks were estimated by running an MCS for 1 million samples. The lifetimes are shown in figures 4.3a-4.3d and serve as the ground reference for measuring the quality of results produced from the MAB algorithms. In the following sections, the results from identifying the top 20, 50 designs out of a set of 100 are shown for each benchmark in detail. For each of the benchmarks, a plot of the sensitivity of the choice of the paremeter ϵ is also shown.

Algorithm 1 MCS algorithm (Uniform sampling)

Let D be the number of designs, n be the sample budget, $n_i = n/D$ where $i \in \{1, ..., D\}$ be the budget per design, m the number of top designs

Input: Vector of designs $\mathbf{D}$

for $D_i \in \mathbf{D}$ **do**
  Get n_i samples of design D_i
  Update the mean $\widehat{\mu}_i$ corresponding to D_i
end for
$\mathbf{I_{sorted}} \leftarrow$ sorted indicies of $\widehat{\mu}_i$ in descending order
$\mathbf{A} \leftarrow \{\mathbf{I_{sorted}}[1], ..., \mathbf{I_{sorted}}[m]\}$

Output: $\mathbf{A}$, the indices for the top m designs

Algorithm 2 SAR algorithm [20]

Let m be desired number of top designs, n be the total sample budget, $k \in \{1, ..., D-1\}$ be the number of phases determined by the number of designs D. The following are also defined:

$$n_k = \left\lceil \frac{1}{\overline{\log}(D)} \frac{n-D}{D+1-k} \right\rceil, \text{ for } k \geq 1 \text{ otherwise } n_k = 0$$

$$\overline{\log}(D) = \frac{1}{2} + \sum_{i=2}^{D} \frac{1}{i}$$

Input: Vector **D** of all designs to evaluate
Setup: Create and operate on a copy of the vector **D**, create an empty vector for accepted designs **A**=[ ], set the number of remaining designs $r = D$, set the number remaining designs to accept $m_{remain} = m$

for each phase $k \in \{1, ..., D-1\}$ **do**
  for each active arm D_i in **D** **do**
    Get $n_k - n_{k-1}$ samples of design D_i
    Update the mean $\widehat{\mu}_i$ corresponding to D_i
  end for
  $[\widehat{\boldsymbol{\mu}}_{\boldsymbol{sorted}}, \mathbf{I}_{\mathbf{sorted}}] \leftarrow$ sort $\widehat{\boldsymbol{\mu}}$ descending
  $\widehat{\Delta}_{upper} = \widehat{\boldsymbol{\mu}}_{\boldsymbol{sorted}}[1] - \widehat{\boldsymbol{\mu}}_{\boldsymbol{sorted}}[m+1]$
  $\widehat{\Delta}_{lower} = \widehat{\boldsymbol{\mu}}_{\boldsymbol{sorted}}[m] - \widehat{\boldsymbol{\mu}}_{\boldsymbol{sorted}}[r]$
  if $\widehat{\Delta}_{upper} > \widehat{\Delta}_{lower}$ **then**
    A.push($\mathbf{I}_{\mathbf{sorted}}[\mathbf{1}]$) ▷ Accept the best design
    D.erase($\mathbf{I}_{\mathbf{sorted}}[\mathbf{1}]$) ▷ Stop sampling this design
    $m_{remain} = m_{remain} - 1$
  else
    D.erase($\mathbf{I}_{\mathbf{sorted}}[\mathbf{r}]$) ▷ Rejected the worst design:
  end if
  $r = r - 1$
end for

Output: **A**, the indices of the top m designs

Algorithm 3 GapE-Var algorithm (Adaptive parameters based on variance)[21]

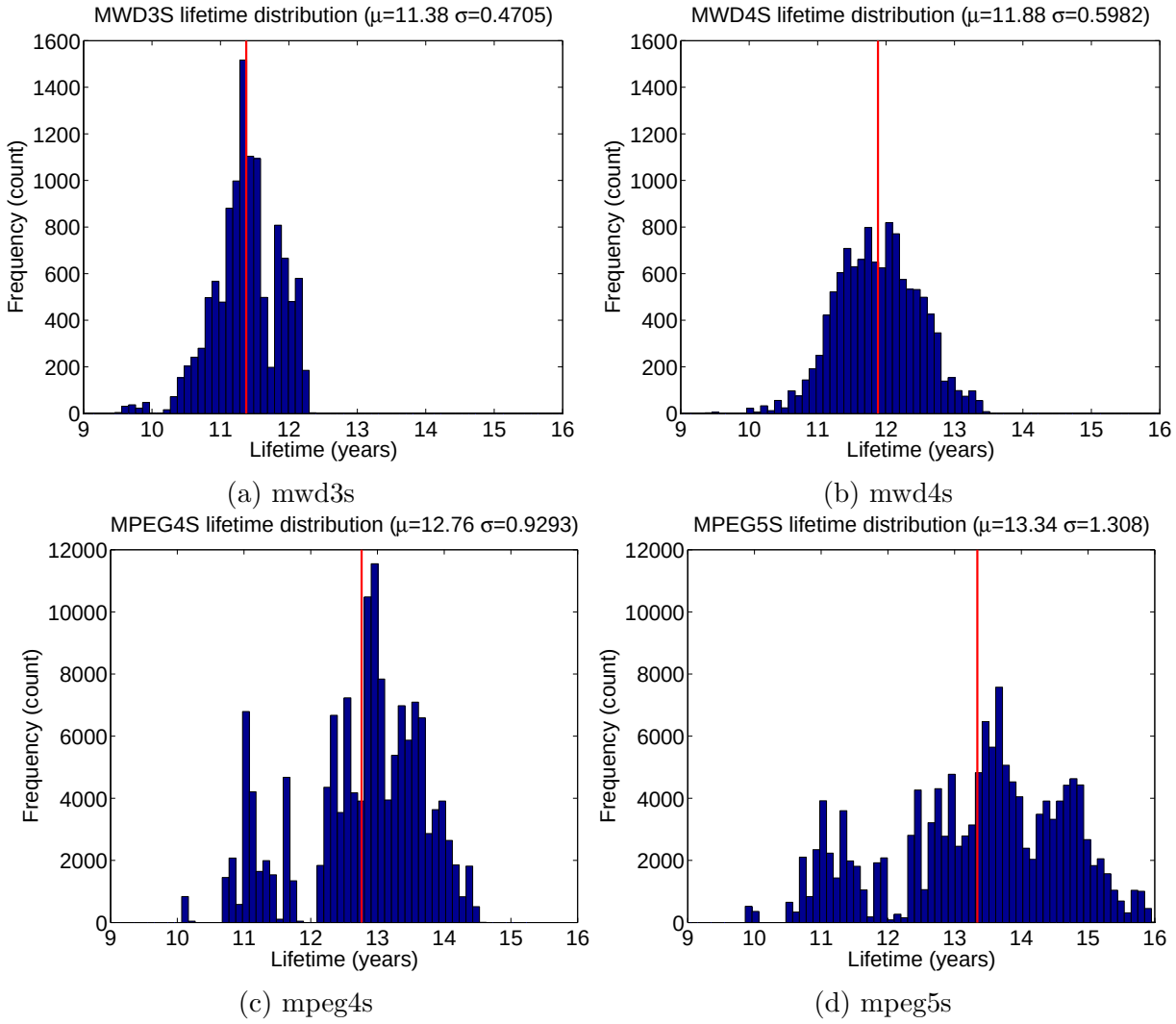
Let m be the desired number of top designs, n be the sample budget, D be the total number of designs, and $[0, b]$ be the bounds of the design lifetime. The following are also defined:

Input: Vector $\mathbf{D}$ of all designs to evaluate

Setup: Initialize by sampling all designs 2 times, update the design means for all the designs $\mathbf{D}_\mu$, compute the empirical gap estimate $\widehat{\mathbf{\Delta}}_m = |\widehat{\mu}_i - \widehat{\mu}_m|$, $i \in \{1, ..., D\}$

for every sample $t = 2D, ..., n$ **do**
  Compute $B_i(t)$ for all D_i in $\mathbf{D}$
  Find the index I which maximizes $\mathbf{B}(t)$
  Sample the design at index I
  Update the mean $\mathbf{D}_\mu[I]$ corresponding to $\mathbf{D}[I]$
  Update the variance $\widehat{\sigma}_I$ corresponding to $\mathbf{D}[I]$
  Recompute $\widehat{\mathbf{\Delta}}_m$
end for
$\mathbf{I}_{\mathbf{sorted}} \leftarrow$ sorted indicies of $\mathbf{D}_\mu$ in descending order
$\mathbf{A} \leftarrow \{\mathbf{I}_{\mathbf{sorted}}[1], ..., \mathbf{I}_{\mathbf{sorted}}[m]\}$

Output: $\mathbf{A}$, the indices of the top m designs

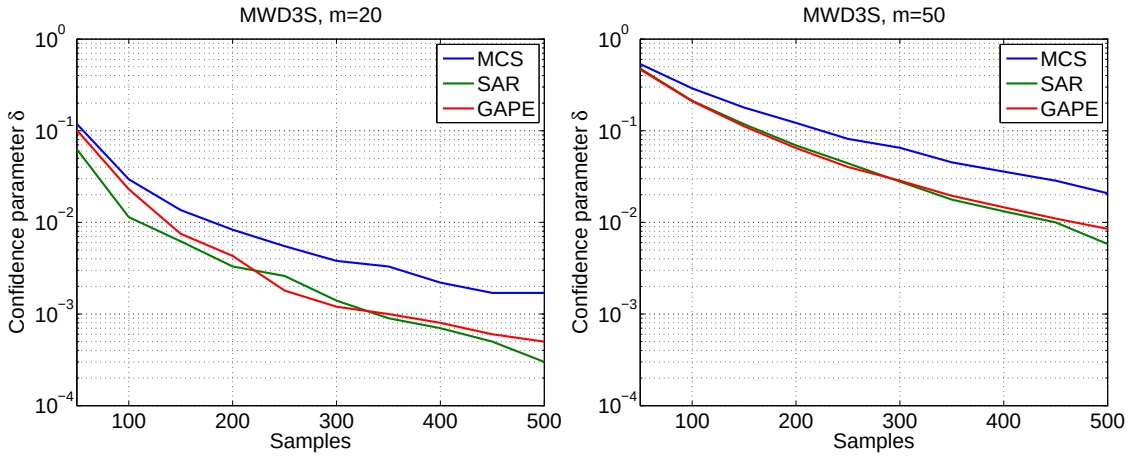


(a) mwd3s (b) mwd4s

(c) mpeg4s (d) mpeg5s

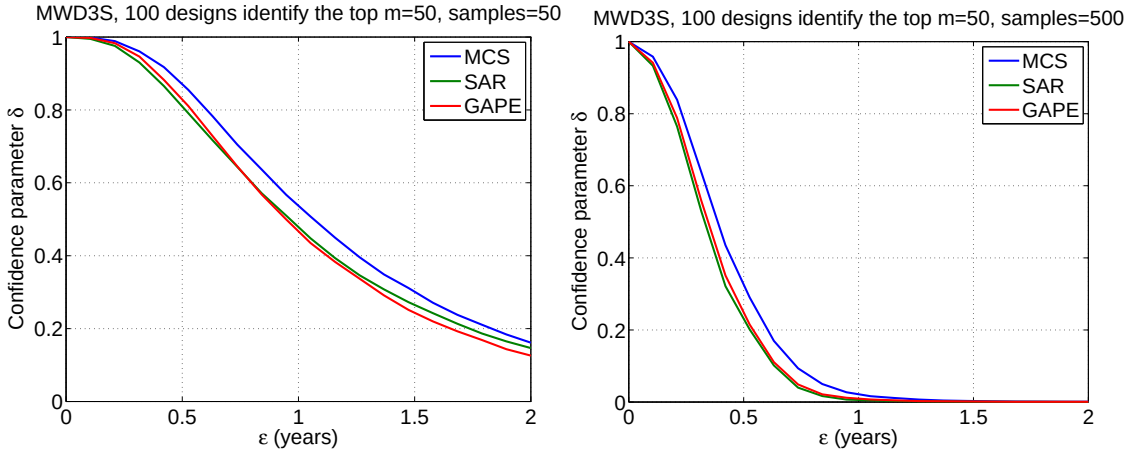
Fig. 4.3 Plots 4.3a-4.3d show the benchmarks actual lifetime distribution which are empirically estimated using one million MCS samples.

4.6.1 MWD3S



(a) MWD3S m=20 (b) MWD3S m=50

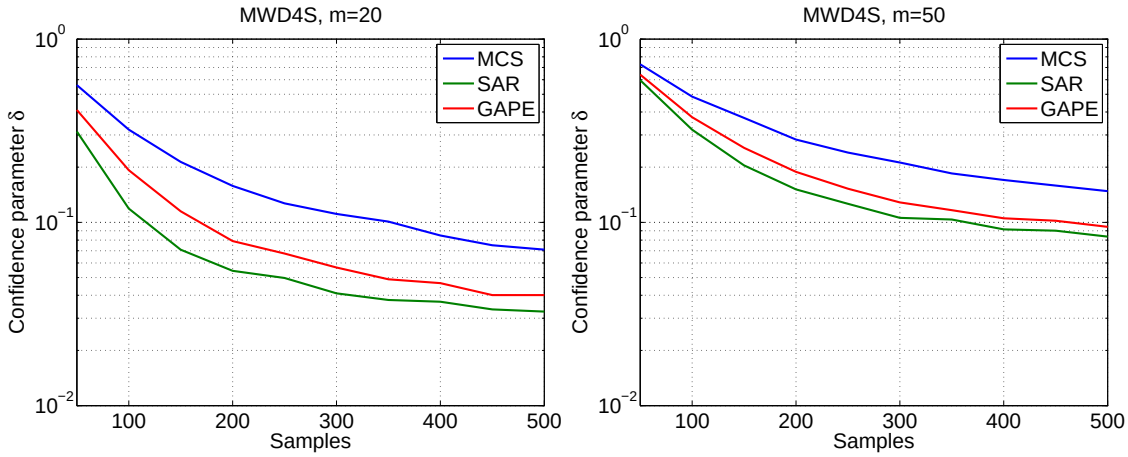
Fig. 4.4 Comparison of MAB against MCS for MWD3S benchmark while picking the top 20, 50 designs out of 100 random designs using $\epsilon = 1$. Plots show the probability of mis-identification over 10,000 trials against the number of samples.



(a) MWD3S 50 samples (b) MWD3S 500 samples

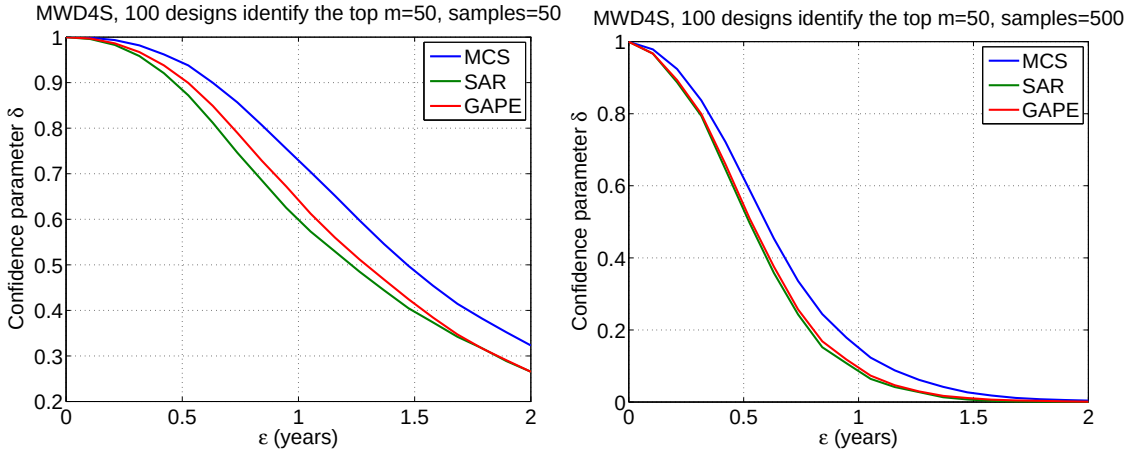
Fig. 4.5 Comparison of MAB against MCS for MWD3S benchmark while picking the top 50 designs out of 100 random designs. Plots show the probability of mis-identification when choosing a different ϵ value for the metric, effects are shown for 50 and 500 samples.

4.6.2 MWD4S



(a) MWD4S m=20 (b) MWD4S m=50

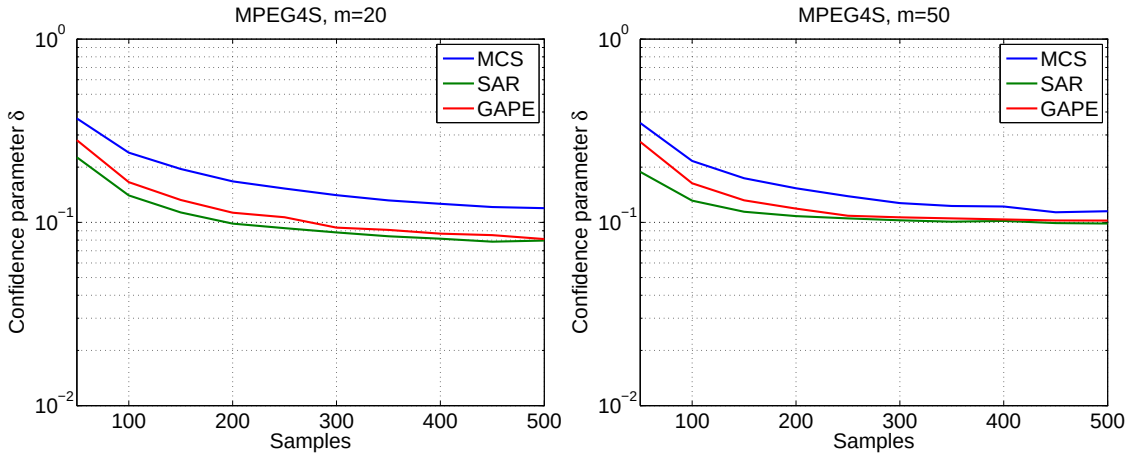
Fig. 4.6 Comparison of MAB against MCS for MWD4S benchmark while picking the top 20, 50 designs out of 100 random designs using $\epsilon = 1$. Plots show the probability of mis-identification over 10,000 trials against the number of samples.



(a) MWD4S 50 samples (b) MWD4S 500 samples

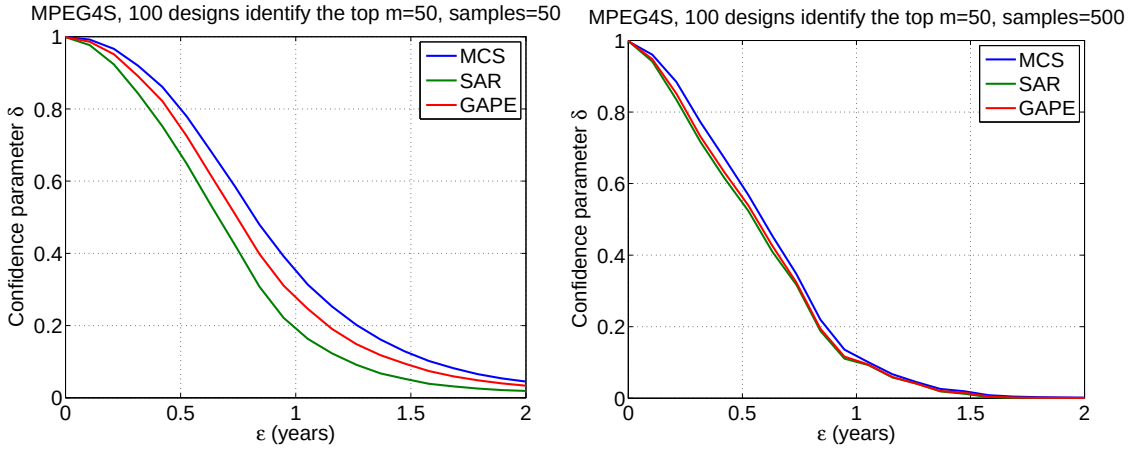
Fig. 4.7 Comparison of MAB against MCS for MWD4S benchmark while picking the top 20, 50 designs out of 100 random designs. Plots show the probability of mis-identification when choosing a different ϵ value for the metric, effects are shown for 50 and 500 samples.

4.6.3 MPEG4S



(a) MPEG4S m=20 (b) MPEG4S m=50

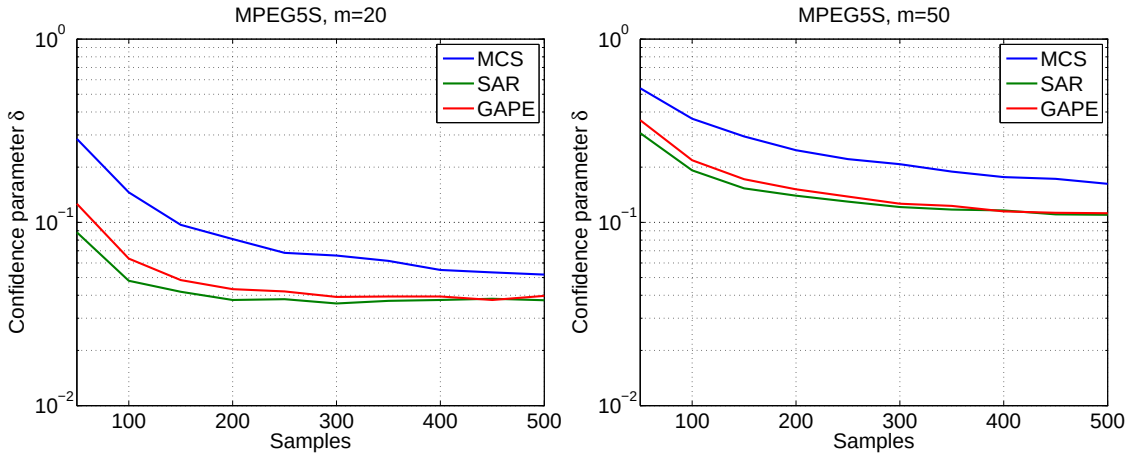
Fig. 4.8 Comparison of MAB against MCS for MPEG4S benchmark while picking the top 20, 50 designs out of 100 random designs using $\epsilon = 1$. Plots show the probability of mis-identification over 10,000 trials against the number of samples.



(a) MPEG4S 50 samples (b) MPEG4S 500 samples

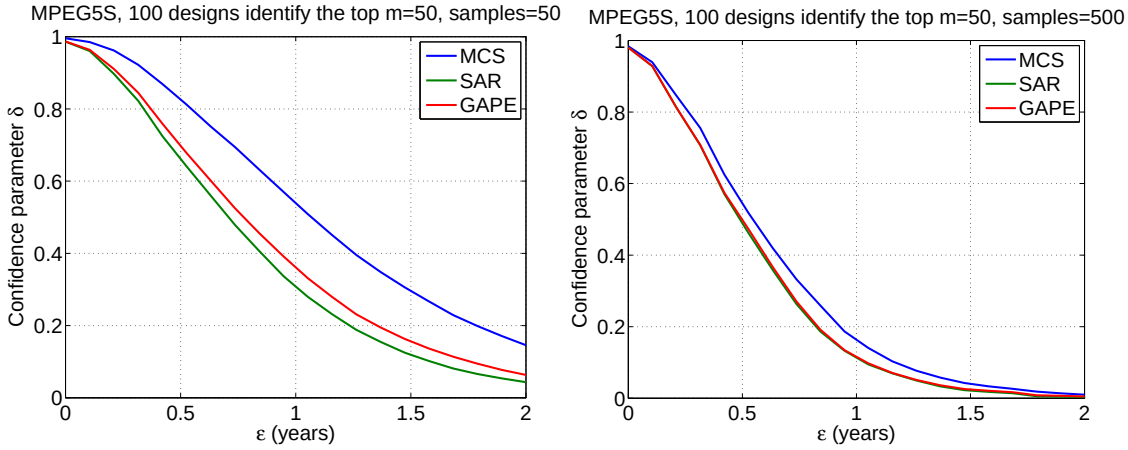
Fig. 4.9 Comparison of MAB against MCS for MPEG4S benchmark while picking the top 20, 50 designs out of 100 random designs. Plots show the probability of mis-identification when choosing a different ϵ value for the metric, effects are shown for 50 and 500 samples.

4.6.4 MPEG5S



(a) MPEG5S m=20 (b) MPEG5S m=50

Fig. 4.10 Comparison of MAB against MCS for MPEG5S benchmark while picking the top 20, 50 designs out of 100 random designs using $\epsilon = 1$. Plots show the probability of mis-identification over 10,000 trials against the number of samples.



(a) MPEG5S 50 samples (b) MPEG5S 500 samples

Fig. 4.11 Comparison of MAB against MCS for MPEG5S benchmark while picking the top 20, 50 designs out of 100 random designs. Plots show the probability of mis-identification when choosing a different ϵ value for the metric, effects are shown for 50 and 500 samples.

The results for selecting the top 20,30,40,50 arms are summarized in table 4.4 which shows amount of savings obtained from using MAB to obtain the same quality answer as MCS could in 500 samples.

4.6.5 Discussion

The results show consistency in the 4 separate benchmarks (figures 4.4a-4.4b, 4.6a-4.6b, 4.8a-4.8b, 4.10a-4.10b); both MAB algorithms are able to outperform the baseline of uniform MCS sampling. The SAR algorithm in general requires even fewer samples than the GapE algorithm to reach a similar confidence δ. In the best case, using the SAR algorithm on the MPEG5S benchmark, the same confidence can be reached in 5.26x fewer samples than the MCS using 500 samples. In the worst case, using the GapE algorithm on the MWD3S benchmark, a sample reduction of 1.45x is seen.

From Table 4.4, the benchmarks in order of most to least improvement are MPEG5S, MPEG4S, MWD4S, MWD3S. By examining the actual design space lifetime-distribution, it is possible to see that the MPEG benchmarks are multi-modal distributed and cover a larger range. This larger spacing and grouping of designs makes the overall identification problem easier and more suited for MAB since the boundary of the 2 sets is more obvious. When the designs are more uni-modal and closer together as is the case with MWD, it is

Table 4.4 Sample savings from MAB algorithms to reach same level of confidence as MCS with 500 samples.

Benchmark	m=20			m=30			m=40			m=50		
	δ	SAR	GapE	δ	SAR	GapE	δ	SAR	GapE	δ	SAR	GapE
MWD3S	0.002	1.92x	1.72x	0.003	1.72x	1.71x	0.009	1.79x	1.67x	0.021	1.49x	1.45x
MWD4S	0.071	3.33x	2.13x	0.112	2.96x	2.07x	0.180	2.54x	2.01x	0.148	2.44x	1.92x
MPEG4S	0.120	3.57x	2.70x	0.101	3.52x	2.48x	0.202	3.60x	2.43x	0.115	3.33x	2.27x
MPEG5S	0.052	5.26x	3.57x	0.083	4.07x	3.05x	0.292	3.70x	3.07x	0.162	3.57x	2.86x

more difficult to separate the sets of designs hence the performance improvement is less noticeable.

For each of the benchmarks, the effects of increasing the size of the top set from $m = 20$ to $m = 50$ generally lead to a lower confidence results. Since the metric for the confidence parameter involves the sum of the regret of all the top m arms, there is cumulatively a larger distance between the aggregated set of lifetimes when the m is larger leading to an overall lower confidence.

In figures 4.4a-4.4b, 4.6a-4.6b ,4.8a-4.8b, 4.10a-4.10b, $\epsilon = 1$ was chosen since it represented roughly a 10% error in lifetime of the set assuming an average lifetime of 10 years. Since the choice of parameter is arbitrary, it is important to understand the effects of the choice. The confidence parameter δ is plotted as a function of ϵ for 50, 500 samples (the end points). From figures 4.5a-4.5b, 4.7a-4.7b, 4.9a-4.9b, 4.11a-4.11b, a decrease in epsilon value generally leads to an increase in error (larger confidence parameter). As expected, increasing the number of samples leads to a lower error (smaller confidence parameter). Irrespective of the ϵ, the MAB still come out ahead with fewer samples.

So far, the basis of the stated improvements have assumed the sampling time is the dominant measure for speedup. This implies the sampling time is much larger than the overhead of the MAB algorithms. If the number of arms to choose from increases, at some point the sampling time will cross over with the overhead time reducing the overall MAB algorithm's effectiveness. A brief analysis is conducted for this implementation of the MAB algorithms. Let D be the number of designs, N be the average number of samples per design, T_{sample} be the time to generate one sample, and $T_{sort}(x)$ be the time to sort the array of x designs, then the time for each algorithm can be roughly approximated as follows (Table 4.5).

We assume that the designs can be sorted in linear time, that is $T_{sort}(D) \rightarrow O(D)$. This

Table 4.5 The run time complexity of each of the algorithms

Algorithm	Description	Run Time (Upper Bound)
MCS	Iterate through each design, samples each design N times	$ND \times T_{sample}$
SAR	The average number of samples per design is the same as MCS. The algorithm linearly sorts all designs for the arm to reject or accept at each phase which happens once per design for a total of D times.	$ND \times T_{sample} + D \times T_{sort}(D)$
GapE	The average number of samples per design is the same. A decision is made at every sample and the algorithm must look through all the designs once to determine which arm to select.	$ND \times T_{sample} + ND \times T_{sort}(D)$

can be accomplished with an insertion sort since the list of designs is partially sorted each time. As the number of designs increase the number of comparisons required to find the next arm to sample increase on the order of D^2 since D decisions need to be made, and each decision requires the designs to be sorted taking D time.

Since number of added samples increases by a constant when the number of designs increases, this grows at a linear rate, $O(D)$. Independent of the type of problem, there is a point where the overhead of the MAB algorithm will overtake the sampling time of the problem since the overhead grows at $O(D^2)$ and the sampling grows at $O(D)$.

For this work, the number of designs does not scale well beyond 100. The GapE algorithm is also not recommended for large number of samples since the complexity is a function of the number of samples as well implying $T(ND^2)$. A size of 100 designs was determined to provide the best overhead time to sampling time trade-off to justify a fair comparison when stating an algorithm uses fewer samples. A summary of the run time for

each of the algorithms at various design set sizes is shown in Table 4.6.

Table 4.6 Time spend evaluate various numbers of designs with an average of 500 samples per design. The time required increases significantly beyond 100 designs. Results collected on an Intel E5-2670, 96GB RAM averaged over 10 trials. Results shown are in seconds.

Algorithm	Number of Designs			
	50	100	200	400
MCS	4.41	8.52	16.86	34.54
SAR	4.48	10.41	27.22	95.26
GapE	5.33	11.46	34.31	108.64

Although MAB has shown promise in identifying the top designs in fewer samples then MCS, one limitation is that it cannot be run on large sets of designs. In the next chapter, we use MAB in a genetic algorithm with a fixed set size of 100 designs; through iteration, the algorithm will converge towards the optimal.

4.7 Future Work

The results collected so far have been on systems where the failure distribution were modeled using log-normal distributions. This was a simplifying assumption since it was difficult to come across a consistent choice of fitting parameters. In reality, a mix of distributions is a more accurate approach and understanding if the results scale for other lifetime models is important in furthering the understanding of MAB algorithm sensitivity.

This work has been limited to 2 bandit algorithms since they natively fit the problem of slack allocation. The works of [20] and [22] have shown the general procedure on how a traditional bandit algorithm can be converted to one which identifies the top set of arms. This procedure can be used on other algorithms but would require further proof that the theoretical bounds are still met.

In recent work on MAB, an algorithm called LILUCB is proposed which allows the top arm to be found bounded following the law of iterated logarithms (LIL); the top arm can be found in *loglog* complexity [31]. While [31] focuses on identifying the single best arm, it provides some useful insight on the budget required for MAB algorithms as well as for reaching a desired confidence. This new algorithm offers a much faster rate of convergence compared with other algorithms (algorithms currently used are *log* bounded). If modified for use in conjunction with lifetime evaluation, this could lead to additional benefits.

Finally, the work so far has only been focused on measuring sample savings in the lifetime objective. This represents the hardest case of the problem since the cost aspect is generally deterministic (if measured as the amount of die area). Since the cost is an important component, the results so far are not as useful for the problem of slack allocation. However, for the problem of finding an optimal task-mapping to maximize lifetime, there is no cost objective, so using this approach would show the clear benefits.

Chapter 5

MAB in Genetic Algorithms

In the previous chapter, we showed that MAB can be used instead of MCS to identify a set of designs with fewer samples for a given confidence level. The problem of slack allocation has still not been fully addressed since the MAB algorithms were unable to prune through a large set of designs effectively. In this chapter, MAB is combined with genetic algorithms so as to converge towards the solution while restricting the maximum number of designs MAB

Genetic algorithms simulate the evolutionary process of natural selection. A population of designs is created and each design is specified in a genetic encoding. Through the process of parent selection, crossover and mutation the population will evolve into a more optimal set. The selection process generally involves an evaluation to determine the *fitness* of the individuals; for the slack allocation problem this would be the design lifetime.

In this chapter we will leverage the results of MAB to reduce the number of samples needed for a fitness evaluation of the population. The principles of the algorithm used in this work can be found in [32].

5.1 Methodology

The basic genetic algorithm (algorithm 4) starts with a randomly selected population of an arbitrary size. The quality of each of the individual designs is determined through an evaluation of the *fitness*; designs with more desirable traits should produce a higher fitness than those without. For each generation, a selection process is used to choose the parents who should reproduce and create new offspring; ideally parents with higher fitness should produce more children. The children are created through a recombination of the genetic encoding of the parents. To create some diversity, each new offspring is mutated with some probability; this helps to prevent the algorithm from getting stuck in a local maximum caused by a poor initial population randomization. Finally, a selection process is used to determine which of the parents and children survive for the next generation. This process is repeated until some end condition is satisfied.

Algorithm 4 Genetic Algorithm [32]

Let **P** be the population of all designs, and G be the number of generations.

Setup: Initialize a population of size p by randomly selecting candidates from **P**, evaluate the designs in the initial population

for every generation $g = 1, ..., G$ **do**
1) Select parents
2) Recombine pairs of parents
3) Mutate the resulting offspring
4) Evaluate new candidates
5) Select individuals for the next generation
end for

Output: Population P_G at the last generation

For most applications, the fitness function is often a deterministic calculation. As such, in step 4, only the new candidates need to be evaluated since the fitness of all previous

designs can be looked up. In lifetime-aware design, the lifetime is estimated using MCS so the accuracy of the fitness will depend on the number of samples used.

One choice of fitness is to simulate each design for a large constant number of samples. In this work, we propose to use MAB in the selection process to eliminate the need to run longer simulations. To accommodate this, algorithm 4 is modified so that MAB is used in step 5. A detailed setup will be given in the sections that follow.

5.1.1 Representation

Each of the designs requires a genetic representation consisting of a string made up of individual genes. The method that is used to encode the slack allocation problem is an integer representation.

Each of the designs are specified with a vector of components $\langle C_1, C_2, ...C_n \rangle$. Depending on the type of the component, they can take on one of many values in a finite integer set. Processors can take on values in the set $\{125, 250, 500\}$ which maps to the number of instructions (in millions) the processors of types Arm M3, Arm7, Arm9 can execute per second. Memories can take on values in the set $\{64, 96, 128, 192, 256, 384, 512, 1024, 2048\}$ which maps to the memory size in kB. As an example, the baseline design in the MWD3S which has 9 processors and 3 memories benchmark is specified with the vector below.

$$\langle 125, 125, 125, 125, 500, 250, 500, 125, 125, 1024, 1024, 1024 \rangle$$

5.1.2 Parent Selection

At each generation, parents are chosen from the population of parents μ to breed; individuals with higher fitness should produce more offspring. The selected parents will breed with a crossover probability p_c, if a selected parent does not breed, then a copy of itself

is placed into the pool of children; these children will still undergo mutation. Some of the most common strategies are as follows.

- Fitness Proportional selection: The probability of selection is proportional to the fitness of an individual
- Rank Proportional Selection: The probability of selection is proportional to the individuals fitness rank
- Tournament Selection: Randomly pick 2 individuals and determine which one gets selected based on some probability (> 0.5 to favours fitter candidates).

In fitness proportional selection, the probability an individual i is selected for mating is a ratio of the individual's fitness and the overall fitness of the parent population $f_i / \sum_{j \in \mu} f_j$. Although this algorithm favours fitter parents, it has the problem that an outstanding individual can dominate the overall selection process not allowing weaker individuals to reproduce and leads to premature convergence. If the fitness values are very close to one another, there is also the problem that all individuals are equally likely to be chosen and there will be insufficient *selection pressure*. This selection method is more sensitive to the fitness function.

To overcome some of the limitations of the fitness proportional selection method, the rank of the individual fitness in the population can be used. This mapping reduces the effects of large and small difference in fitness and as a result the selection pressure is more constant. There are 2 common schemes for transforming the ranks into selection probabilities. The selection probability can decrease linearly from the fittest design or if higher selection pressure is desired, an exponential decrease can be used.

In the previous strategies, the selection process has depended on the accuracy of the fitness function for the parent population μ; in problems where it is more computationally

expensive to accurately evaluate the fitness for entire population, tournament selection can be used. Tournament selection is best used where it is difficult to get an exact fitness value for an individual and where comparison of two individuals is an easier problem. In this selection process, 2 random individuals are chosen and with some probability p, the fitter individual gets chosen. Tournaments can also be held with more individual but there will be a stronger selection pressure since weaker individuals are more unlikely to be chosen.

5.1.3 Recombination

Recombination is the process of creating new individuals from parent solutions. For simplicity, we will assume that 2 parents are selected and produce 2 new offspring although the process can be generalized to multiple parents. The valid crossover strategies for this problem are: 1-point, N-point, Uniform crossover.

The process of creating new children involves the genetic representation of the parents being split and spliced together with the tail sections swapped. For a simple 1-point recombination, a random value between $[0, l]$, where l is the length of the genetic representation is chosen as the splitting and splicing point. This can be generalized to N random splitting points for the N-point strategy. In the uniform crossover scheme, with some probability p, the gene will be taken from the first parent otherwise it will be taken from the second parent; this process iterates through all the genes in the genetic representation.

The choice of scheme has been widely studied and will lead to biases. Both 1-point and N-point crossover favour keeping genes near the ends together. In N-point crossover, if N is odd, then there is a strong *positional bias* against the ends of the genotype being from the same parent. Although uniform crossover does not have a positional bias it has a *distribution bias*. If a probability of 0.5 is chosen, then equal amounts of genetic components are taken from both parents; the genetic makeup of the offspring will favour neither of the

parents.

5.1.4 Mutation

The purpose of mutation is introduces some genetic variations into the offspring. Consider the extreme example, where the cross over probability $p_c = 0$. Since no offspring are generated from crossover, the entire next generation will only be made up of mutations.

For integer genetic algorithms, there are 2 types of mutation which can occur on the: random resetting and creep mutation. Random resetting refers to a flip in the state of one of the genes with some mutation probability p_m. Creep mutation manifests as an increment or decrements to the gene and works best if the integer problem is ordinal. The amount of creep can be controlled by choosing different sampling distributions.

5.1.5 Survivor Selection

In genetic algorithms, it is often desired that the population size remain constant or grow at a constrained rate throughout the generations. Since the number of parents μ and number of offspring λ often exceeds the population constraint after reproduction, the individuals which survive to the next generation must be selected. Here are a few possible selection policies:

- Age based replacement: The previous μ parents die and the λ offspring form the new generation in a FIFO fashion
- Fitness based replacement: Some of the parents and children survive depending on the fitnesses
- Replace worst: The worst λ parents are selected for replacement

The main disadvantage of using age based replacement over fitness based replacement is that it is often easy to lose the best member of the population. This problem can be mitigated by using elitism: a select percentage of the top individuals are kept alive unchanged in the next generation. For fitness based replacements, similar strategies to the parent selection can be used (i.e fitness proportionate or tournament selection).

The strategy to replace the worst of the of the parents can lead to rapid improvements to the overall fitness but often reduces population diversity and can lead to premature convergence. Often this policy is used with a check to ensure that that there are no duplicate individuals in the population to maintain an artificial diversity.

MAB selection

In this work, MAB is proposed to assist in the selection process of the next generation. In the overall process of the GA, there are μ parents who will create λ offspring. Using MAB, the top μ individuals are found from the aggregate set of the parents and new children $\mu + \lambda$. In effect, this method is most similar to the *replace worst* and the use of MAB acts as a mean to determine the worst set without deterministic fitness values.

5.2 Experimental Setup

This section outlines the setup of the genetic algorithm where MAB is used as a selection strategy. A initial population size of $\mu = 100$ was chosen. The number of offspring to produced at each stage is $\lambda = 100$ generated using tournament selection with probability $p = 1.0$ of selecting the fitter design. All the children are created using a 1-point crossover policy ($p_c = 1.0$, no children created purely from mutations). Each child has a mutation probability of $p_m = 0.1$ and a random resetting policy is used. Assuming an average gene

length of 10, this roughly represents a flip of one of the genes out of every children. Finally the proposed MAB selection strategy is used to find the next generation composition. This is used in conjunction with a no duplication policy in the population. The details for this genetic algorithm are summarized in table 5.1.

Table 5.1 An outline of criteria for a GA setup for the slack allocation problem

Representation	Integer
Population Size	100
Initialization	Random 100
Parent selection	Tournament $p = 1$
Recombination	1-Point Crossover $p_c = 1$
Mutation	Random resetting $p_m = 0.1$
Survival selection	Replace worst (MAB)
End Condition	Run 50 generations

The results of the GA are collected over 100 separate trials and averaged for each setup. Since the purpose is to understand the improvements that the MAB can provide, various levels of sample budget were allocated for each of the algorithms. For the MCS algorithm results are shown for: 1, 10, 100 average samples per design. For the MAB algorithms, results are shown for 10, 100 average samples per design.

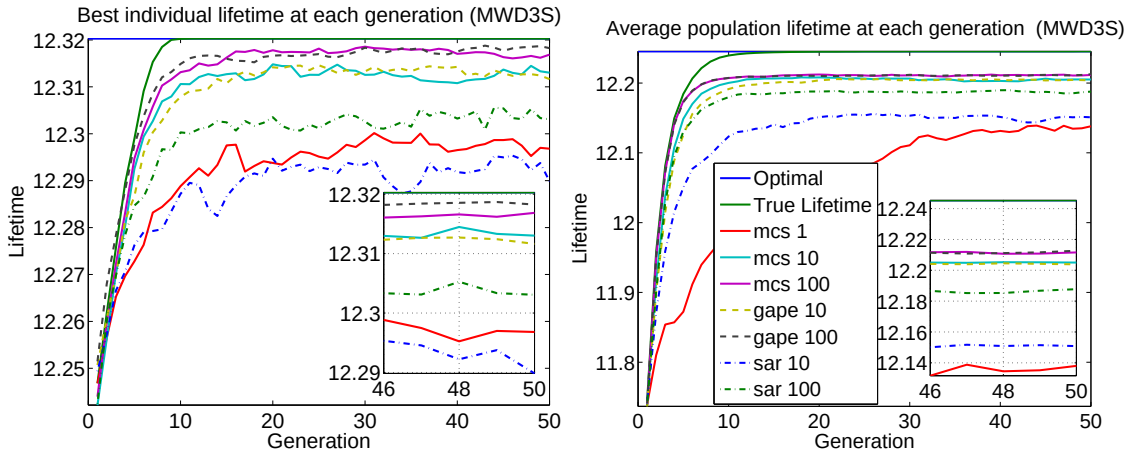
Metrics

The metrics which are use to quantify the quality of the solutions are: the best fitness and the mean fitness of the population reached at the end of the simulation (generation 50).

5.3 Results

The optimal lifetime discovered from GA is shown for each benchmark. A table of the final lifetime and variance is also included.

5.3.1 MWD3S



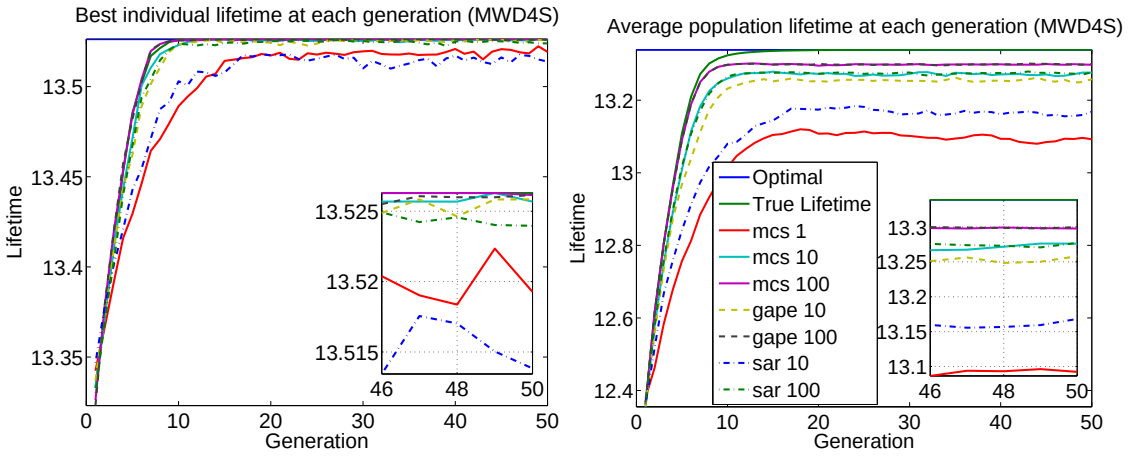
(a) Best design lifetime (b) Average design lifetime

Fig. 5.1 Quality of the best design found at each generation from the genetic algorithm. Sub-axis shows a zoomed in plot of the last generations.

Table 5.2 The best lifetime and average population lifetime of the top set at the last generation. The samples represent an average number of samples per design since SAR and GapE sample non uniformly. Lifetimes are shown with 1 standard deviation over 100 trials for MWD3S benchmark.

Algorithm	Samples	Best fitness	Mean population fitness
MCS	1	12.2968 ± 0.0160	12.1380 ± 0.0746
	10	12.3130 ± 0.0121	12.2050 ± 0.0122
	100	12.3169 ± 0.0086	12.2116 ± 0.0082
GapE	10	12.3116 ± 0.0120	12.2039 ± 0.0115
	100	12.3182 ± 0.0061	12.2127 ± 0.0061
SAR	10	12.2898 ± 0.0223	12.1509 ± 0.0342
	100	12.3031 ± 0.0190	12.1877 ± 0.0140

5.3.2 MWD4S



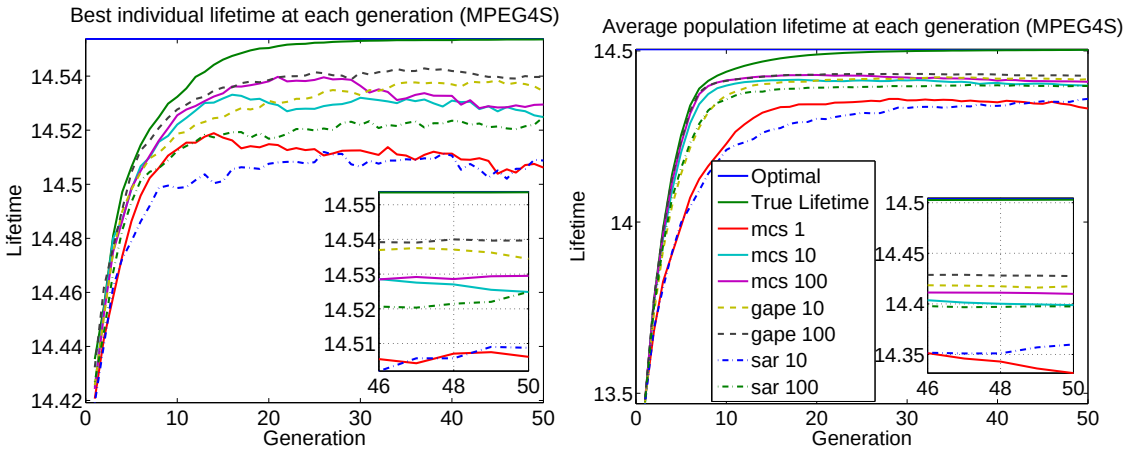
(a) Best design lifetime (b) Average design lifetime

Fig. 5.2 Quality of the best design found at each generation from the genetic algorithm. Sub-axis shows a zoomed in plot of the last generations.

Table 5.3 The best lifetime and average population lifetime of the top set at the last generation. The samples represent an average number of samples per design since SAR and GapE sample non uniformly. Lifetimes are shown with 1 standard deviation over 100 trials for MWD4S benchmark.

Algorithm	Samples	Best fitness	Mean population fitness
MCS	1	13.5193 ± 0.0215	13.0924 ± 0.0764
	10	13.5257 ± 0.0036	13.2761 ± 0.0248
	100	13.5261 ± 0.0015	13.2978 ± 0.0109
GapE	10	13.5258 ± 0.0026	13.2581 ± 0.0323
	100	13.5261 ± 0.0015	13.2998 ± 0.0089
SAR	10	13.5138 ± 0.0300	13.1682 ± 0.0762
	100	13.5240 ± 0.0095	13.2771 ± 0.0210

5.3.3 MPEG4S



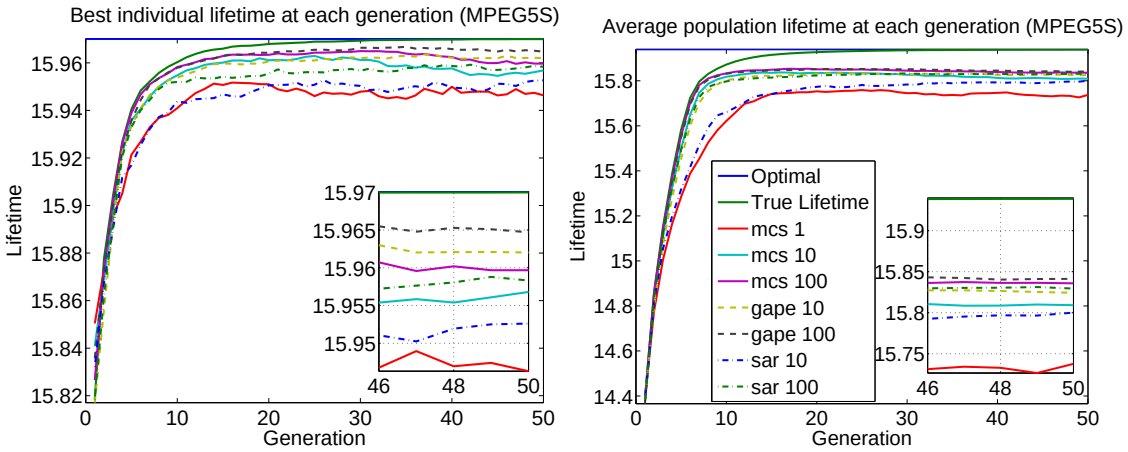
(a) Best design lifetime (b) Average design lifetime

Fig. 5.3 Quality of the best design found at each generation from the genetic algorithm. Sub-axis shows a zoomed in plot of the last generations.

Table 5.4 The best lifetime and average population lifetime of the top set at the last generation. The samples represent an average number of samples per design since SAR and GapE sample non uniformly. Lifetimes are shown with 1 standard deviation over 100 trials for MPEG4S benchmark.

Algorithm	Samples	Best fitness	Mean population fitness
MCS	1	14.5062 ± 0.0241	14.3316 ± 0.0551
	10	14.5249 ± 0.0195	14.3988 ± 0.0239
	100	14.5295 ± 0.0175	14.4099 ± 0.0140
GapE	10	14.5344 ± 0.0202	14.4175 ± 0.0212
	100	14.5397 ± 0.0159	14.4275 ± 0.0177
SAR	10	14.5088 ± 0.0192	14.3599 ± 0.0352
	100	14.5249 ± 0.0177	14.3974 ± 0.0128

5.3.4 MPEG5S



(a) Best design lifetime (b) Average design lifetime

Fig. 5.4 Quality of the best design found at each generation from the genetic algorithm. Sub-axis shows a zoomed in plot of the last generations.

Table 5.5 The best lifetime and average population lifetime of the top set at the last generation. The samples represent an average number of samples per design since SAR and GapE sample non uniformly. Lifetimes are shown with 1 standard deviation over 100 trials for MPEG5S benchmark.

Algorithm	Samples	Best fitness	Mean population fitness
MCS	1	15.9463 ± 0.0192	15.7377 ± 0.0587
	10	15.9568 ± 0.0093	15.8092 ± 0.0292
	100	15.9597 ± 0.0076	15.8358 ± 0.0208
GapE	10	15.9620 ± 0.0091	15.8243 ± 0.0265
	100	15.9647 ± 0.0073	15.8413 ± 0.0225
SAR	10	15.9526 ± 0.0099	15.8001 ± 0.0219
	100	15.9583 ± 0.0087	15.8294 ± 0.0206

5.3.5 Discussion

For each of the benchmarks, the GA was used to find the design with the highest lifetime. The effectiveness of the MAB algorithms are compared for 10 and 100 average samples per

design. As a point of reference, the performance of the GA is shown when the true lifetimes are known and when only 1 sample is used per design. Between each of the generations, the results are saved and loaded for subsequent generations.

For the problem of identifying the best group of designs, the average lifetime of the population at each generation is examined; in this case, the MAB do not outperform the MCS approach. Similar quality of answers is obtained for GAPE 10 samples and MCS 10 samples for, and comparable results for GAPE 100 and MCS 100. SAR consistently under-performs with SAR 100 performing worst than MCS 10 samples.

The previous results presented in chapter 4 may have been incomplete since they were based around assumptions that may no longer be valid. Firstly, the results were presented on the basis that the designs were randomly chosen from the design space; in the case of GA, these designs are no longer random since they depend on previous generations. The designs tend to converge closer and closer together and do not represent a good sample of the lifetime distribution. This convergence to a near uniform lifetime distribution is a more difficult problem than the initial random problem since the lifetime gap spacing and range have been reduced.

The metric used in chapter 4 to measure performance was the confidence parameter δ with an associated ϵ. In genetic algorithms, when the problem has nearly converged, making the correct decision implies a much smaller ϵ value making the previously assumed $\epsilon = 1$ invalid. From the standard deviation in the mean population fitness (depending on the benchmark), a confidence of 3σ is generally within ± 0.1 years. If designs are to be distinguished within $\epsilon = 0.1$, then many more samples will be required. Extrapolating from 4.4a-4.4b, 4.6a-4.6b ,4.8a-4.8b, 4.10a-4.10b, as the samples are increased, the near convergence behavior will result little difference between the MAB and MCS algorithms.

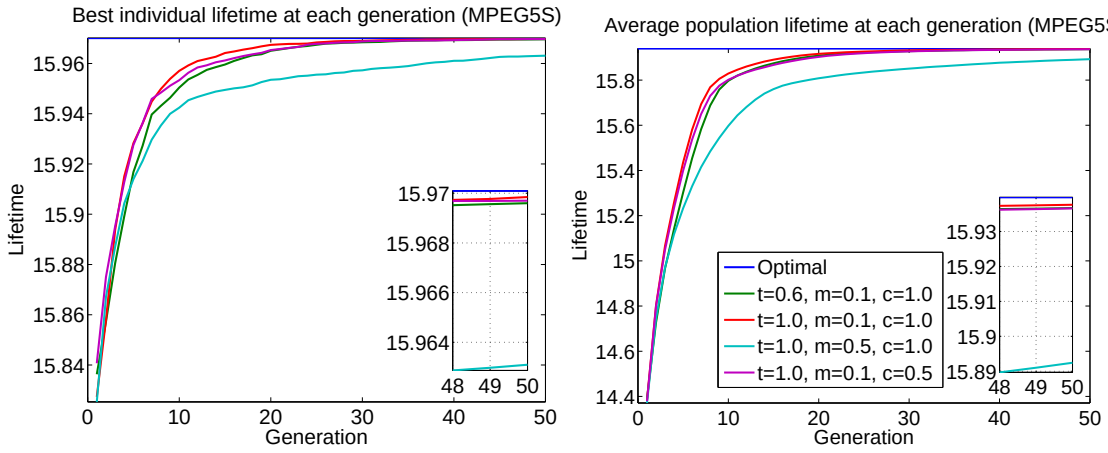
Another consideration is that the whole sampling process is sensitive to estimation

noise of the lifetime since new designs are estimated with few samples. These inaccuracies are enough to cause the GA algorithms to converge to a sub-optimal answer leading to discrepancies not exceeding 0.06 years for the best design lifetime and 0.25 years for the average design lifetime. Later in this chapter, in figure 5.5b and figure 5.5a, it is shown that a design much closer to the optimal is found when the true lifetimes are known. The estimation noise also makes it possible to lose good designs and hence the results as the generations progress will not necessarily be monotonically increasing.

In the problem of identifying the best arm/design rather than the population average, the GAPE 10 samples is able to outperform the MCS 100 only in the MPEG benchmarks. The results should however still be interpreted with caution since, the actual difference in lifetime within 0.01 (a conservative estimate). This represents a lifetime estimate accurate to within 3.65 days for a device deemed to last around 10 years. At this point, the modeling inaccuracies become dominant and although the GAPE is able to outperform the MCS, the practical significance is minimal.

The effect of saving samples between generations will affect the outcome of the chosen set of designs for the next generation. Designs which have been previously sampled will have lower variance than any new designs which are introduced. In general this has minimal effect on MCS since the variance of each of the designs is approximately uniform. For the MAB algorithm, the designs around the decision boundary will have the lowest variance. Since GAPE algorithm takes into account the variance of the designs that it has sampled, it can strategically continue to sample unknown designs that have been generated at the next iteration. The under-performance of SAR is most likely attributed to the fixed phase sampling strategy. Since SAR ignore all previous information and only relies on the current lifetime means, it is more likely that a sub-optimal arm will appear to be a better solution than one which has a lower variance and result in a incorrect decision.

As a final remark, the GA was tested for sensitivity to parameter changes on the MPEG5S benchmark as shown in figure 5.5a-5.5b (this benchmark showed the most promising results). A variety of values for tournament, mutation, crossover probabilities were chosen. The parameters for optimal performance in terms of the speed of convergence was found to be using deterministic tournaments ($t = 1$), all children produced by crossover ($t_c = 1$) and a mutation rate of ($t_m = 0.1$). These optimal values were used in the previous simulation but in the event that a better parameter choice exist, the effects are expected to be minimal.



(a) Best design lifetime (b) Average design lifetime

Fig. 5.5 Quality of the best design found at each generation from the genetic algorithm using different GA parameters. Sub-axis shows a zoomed in plot of the last generations.

5.4 Extension

MAB have been used to examine the single objective problem of lifetime optimization in the slack allocation problem. However, the problem of slack allocation is a multi-objective problem since it is meaningless to talk of lifetime improvements while ignoring the cost. In this section, the single objective approach is extended to account for both the cost and

lifetime.

The two objectives are combined using a weight so that it can be used in the single objective optimizer (5.1).

$$(\alpha)\text{Lifetime} - (1-\alpha)(\text{Cost}) \tag{5.1}$$

By sweeping through a range of weights (α) it is possible to find the Pareto optimal designs assuming design space is convex; this method of scalarization will be refereed to as the *weighted sums* method. Since the cost of an MPSoC is difficult to estimate, the die area obtained from floorplanning is used as an approximation.

The design space is shown for each of the benchmarks as well as with their respective Pareto optimal points is shown for reference in figures 5.6a-5.6d. Due to the actual shape of the design space, some Pareto optimal solutions cannot be found. In general only 18-52% of the Pareto optimal designs can be found using the weighted sums method; the exact number of Pareto optimal points which exist and which can be found is shown in table 5.6.

Table 5.6 The number of Pareto optimal designs per benchmark as well as the number which can be discovered using weighted sums.

Benchmark	# of Pareto optimal designs	# Pareto optimal designs (weighted sums)
MWD3S	19	10
MWD4S	23	11
MPEG4S	43	8
MPEG5S	47	10

Given a set of points S, we want to compare the distance to the Pareto optimal set R. One metric to measure the closeness of the solution is the *average distance from reference set* (ADRS) given by [33] as follows.

$$\text{ADRS}(R,S) = \frac{1}{|R|}\sum_{i=1}^{|R|}\left(\min_{s\in S}\{c(r_i,s)\}\right) \tag{5.2}$$

where

$$c(r_i, s) = \max_{j=1,2,\ldots,M} \left\{0, w_j\big(f_j(s) - f_j(r_i)\big)\right\} \tag{5.3}$$

and w_j is the reciprocal of the range of objective f_j in the reference set R. This metric is a normalized measure which finds the minimum distance of a point with respect to the worst objective.

The reference set which is used for comparison is the optimal set of points which can be found by the weighted sums approach. For the purposes of comparing algorithms, only the weights which will result in the discovery of a unique Pareto optimal point are explored. For each of these weights, the best design in the last generation is taken as the optimal point.

The results of the ADRS averaged over 30 trials are shown in figures 5.7a-5.7b for when 10 and 100 average samples per design are used. The average ADRS over all the benchmarks is given in table 5.7. The results show that on average, GapE performs better than SAR for both 10 and 100 samples across all benchmarks. For 10 samples, GapE has an ADRS lower by 0.0051, and for 100 samples, GapE has an ADRS lower by 0.0040. These results remain consistent with the results from the single objective situation. However, a contradictory result is that MCS consistently attains a lower ADRS compared to either of the MAB algorithms. To further understand this, we need to look at the discovered Pareto optimal front.

In order to understand the significance of the ARDS differences, it is helpful to look at the Pareto optimal fronts. The Pareto optimal fronts for each benchmark are plotted in 5.8a-5.8d for 100 average samples per design to show the closeness of solutions. Even though the ADRS values are different, for practical purposes the differences do not manifest in a significant way.

Table 5.7 The ADRS averaged across all benchmarks.

Algorithm	Samples	ADRS
MCS	1	0.1009
	10	0.0622
	100	0.0511
GapE	10	0.0684
	100	0.0542
SAR	10	0.0735
	100	0.0582

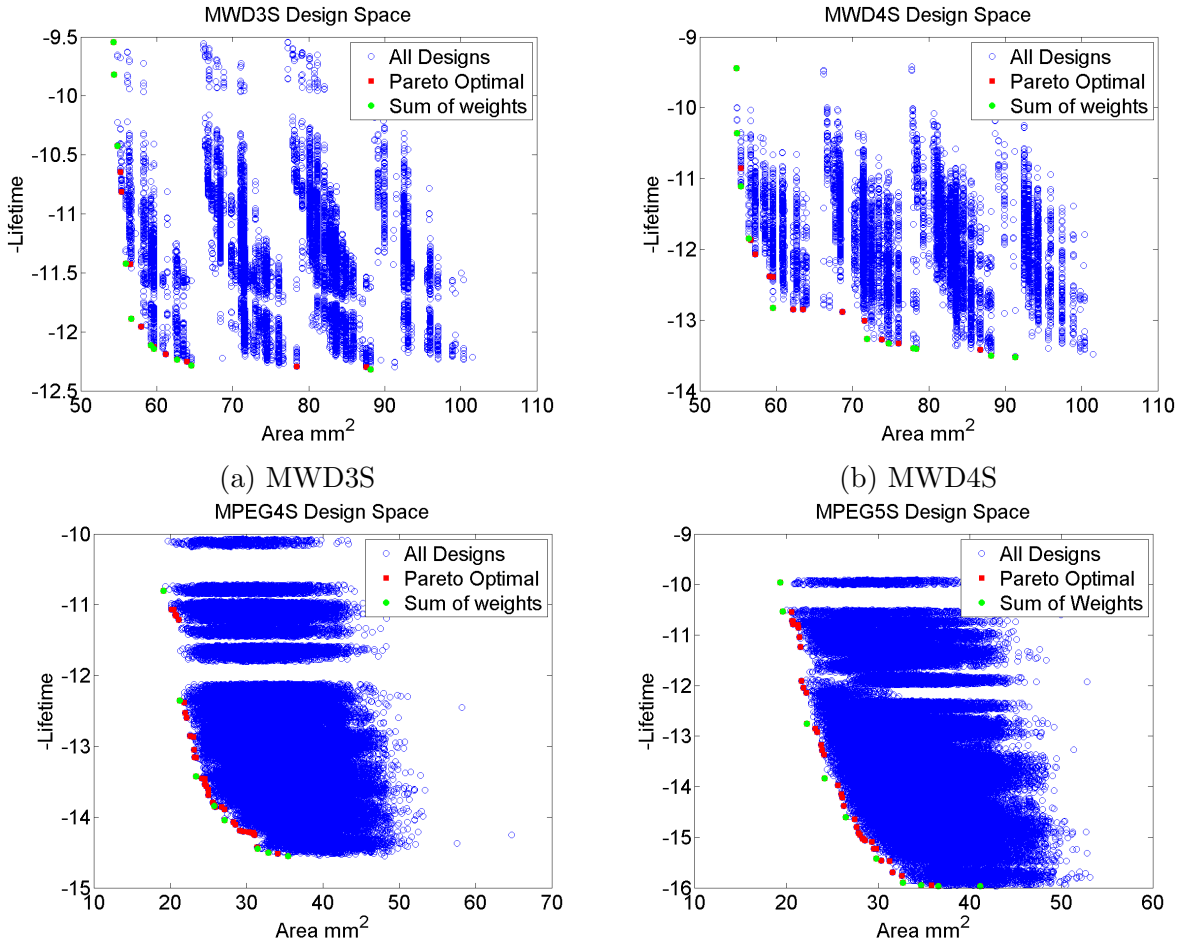
From these graphs it is difficult to quantify the practical benefits of achieving a small ADRS improvement. These plots show that using a single MCS sample get an answer very similar to when 100 samples are used.

Previously, it was determined that using MAB in GA can lead to some improvements in finding the optimal lifetime but these improvements are small. Taking into account the fact that the area estimate for each design is predetermined, the points on the Pareto optimal front which involve a higher cost weighting are more easily estimated. When looking at the Pareto optimal fronts in figures 5.8a-5.8d, the method fails to find the highest lifetime point for 3 out of the 4 benchmarks. The only benchmark for which this point is identified is in the MPEG5S which has been shown to be the easiest benchmark typically yielding the highest improvement.

5.5 Future Work

The area of lifetime-aware design is an open area for research with many heuristic approaches. This chapter has presented a single objective genetic algorithm with MAB integrated as a selection strategy. To solve the multi-objective problem, a weighted sum is

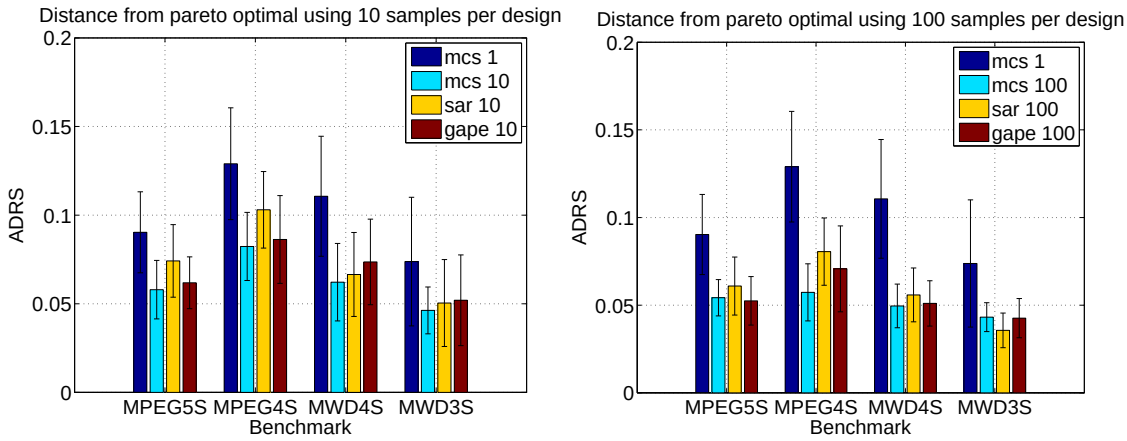
used. In general there are better ways to solve integer multi-objective optimization which do not involve scalarization. Since the design space is non-convex, other algorithms may also be able to remedy the problem that some solutions cannot be discovered from using a weighted sum.



(a) MWD3S (b) MWD4S

(c) MPEG4S (d) MPEG5S

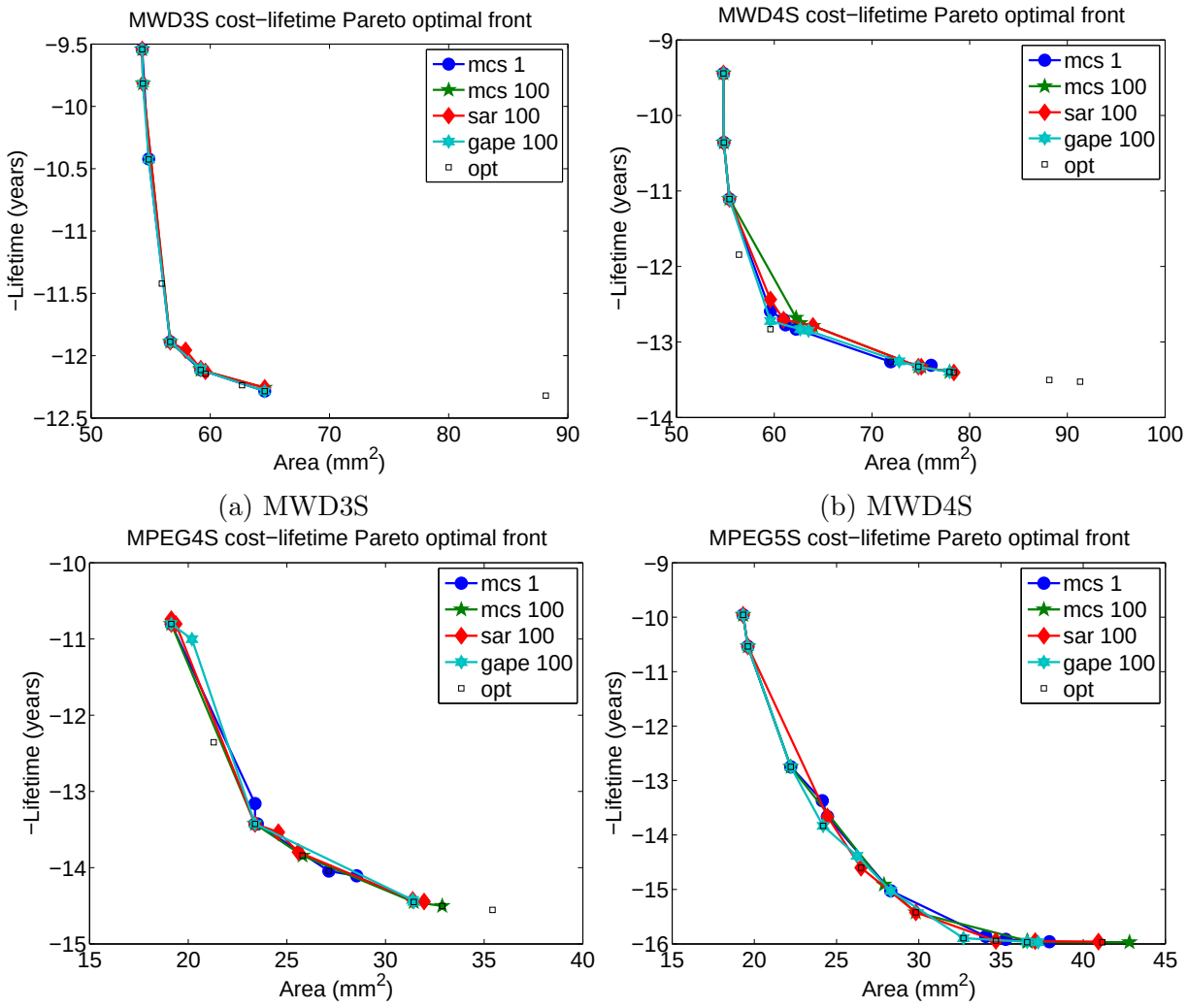
Fig. 5.6 Plots 5.6a-5.6d show all the design points as well as the cost to lifetime Pareto optimal front. The designs which can be found by the sum of weights method are also shown.



(a) 10 samples per design

(b) 100 samples per design

Fig. 5.7 ADRS across all benchmarks using MAB at 10, 100 average samples per design respectively. Error bars are showing 1 standard deviation.



(a) MWD3S (b) MWD4S

(c) MPEG4S (d) MPEG5S

Fig. 5.8 Plots 5.6a-5.6d show all the design points as well as the cost to lifetime Pareto optimal front. The designs which can be found by the sum of weights method are also shown.

Chapter 6

Conclusion

In the problem of identifying the optimal allocation of slack to an initial MPSoC design, multi-armed bandits was proposed as an alternate method to MCS for discovering the best designs in a larger set of designs. It was shown for a random set of 100 designs, MAB finds the best designs between 1.45x-5.26x times fewer samples. Due to the scalability problem, MAB could not effectively explore the entire design space; for this MAB was used in a genetic algorithm.

MAB was used as a selection algorithm in a GA. Each of the designs was represented using a genetic encoding. By doing random crossovers and mutations on these genetic representations, it was shown that the GA converges to a solution for the slack allocation problem. The GA results showed there is no improvement in the average population lifetime in when the average samples per design is kept constant. For the problem of identifying the top design, the MAB was able to find a design 0.01 years better than the equivalent Monte-Carlo Simulation in one benchmark (MPEG).

Using a weighted combination of the cost and lifetime objectives, it was possible to discover the 18-52% of the Pareto optimal designs depending on the benchmark. When

using a single sample to estimate the lifetime of the design, it is possible to find the Pareto optimal front with an ADRS distance metric of within 0.1009. This figure improves to 0.0622 when up to 10 samples are used.

MAB are effective at determining the best design out of a set which follows a more typical lifetime distribution. MAB has difficulty discriminating between designs after they have converged closely to a more optimal set when used in GA. In summary, the use of MAB algorithms for design space exploration should be applied with caution.

References

[1] A. S. Hartman, D. E. Thomas, and B. H. Meyer, "A case for lifetime-aware task mapping in embedded chip multiprocessors," *CODES+ISSS*, pp. 145–154, 2010.

[2] L. H. L. Huang, F. Y. F. Yuan, and Q. X. Q. Xu, "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," *DATE*, pp. 51–56, 2009.

[3] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "Lifetime reliability: Toward an architectural solution," *IEEE Micro*, vol. 25, no. 3, pp. 70–80, 2005.

[4] B. Nahar and B. Meyer, "Rotr: Rotational redundant task mapping for fail-operational mpsocs," in *DFTS*, pp. 21–28, Oct 2015.

[5] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "Exploiting structural duplication for lifetime reliability enhancement," *ISCA*, pp. 520–531, 2005.

[6] B. H. Meyer, A. S. Hartman, and D. E. Thomas, "Slack allocation for yield improvement in NoC-based MPSoCs," *ISQED*, pp. 738–746, 2010.

[7] J.-Y. Audibert and S. Bubeck, "Best Arm Identification in Multi-Armed Bandits," in *COLT*, (Haifa, Israel), p. 13, 2010.

[8] T. L. Lai and H. Robbins, "Asymptotically Efficient Adaptive Allocation Rules," *Adv. Appl. Math.*, vol. 6, pp. 4–22, 1985.

[9] JEDEC Solid State Technology, "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122-B*, 2003.

[10] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The case for lifetime reliability-aware microprocessors," *ISCA*, 2004.

[11] S. Arunachalam, T. Chantem, R. P. Dick, and X. S. Hu, "An online wear state monitoring methodology for off-the-shelf embedded processors," *CODES+ISSS*, pp. 114–123, 2015.

[12] Z. Gu, C. Zhu, L. Shang, and R. P. Dick, "Application-specific MPSoC reliability optimization," *VLSI Systems*, vol. 16, no. 5, pp. 603–608, 2008.

[13] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, L. Shang, A. Arbor, and N. Dame, "System-level reliability modeling for MPSoCs," in *CODES+ISSS IEEE/ACM/IFIP*, pp. 297–306, Oct 2010.

[14] A. S. Hartman and D. E. Thomas, "Lifetime improvement through runtime wear-based task mapping," *CODES+ISS IEEE/ACM/IFIPS*, p. 13, 2012.

[15] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MP-SoCs," *DATE*, pp. 1–6, 2014.

[16] W. R. Thompson, "On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[17] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin of the American Mathematical Society*, vol. 58, no. 5, pp. 527–536, 1952.

[18] R. E. Bellman, "The Theory of Dynamic Programming," *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954.

[19] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2/3, pp. 235–256, 2002.

[20] S. Bubeck, T. Wang, and N. Viswanathan, "Multiple identifications in multi-armed bandits," *ICML*, vol. 28, pp. 258–265, 2013.

[21] V. Gabillon, M. Ghavamzadeh, A. Lazaric, and S. Bubeck, "Multi-Bandit Best Arm Identification," in *Advances in Neural Information Processing Systems 24* (J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, eds.), pp. 2222–2230, Curran Associates, Inc., 2011.

[22] V. Gabillon, M. Ghavamzadeh, and A. Lazaric, "Best arm identification: A unified approach to fixed budget and fixed confidence," *Advances in Neural Information Processing Systems (NIPS)*, vol. 4, pp. 3212–3220, 2012.

[23] D. J. Sorin, "Fault Tolerant Computer Architecture," in *Synthesis Lectures on Computer Architecture*, vol. 4, pp. 1–104, 2009.

[24] JEDEC Solid State Technology, "Methods for Calculating Failure Rates in Units of FITs," *JEDEC Publication JEP122-A*, 2001.

[25] J. W. McPherson, *Reliability Physics and Engineering Time-To-Failure Modeling*. Boston, MA: Springer Science+Business Media, LLC, 2010.

[26] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *VLSI*, vol. 11, no. 6, pp. 1120–1135, 2003.

[27] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," *ISCA*, p. 2, 2003.

[28] E. Jaspers and P. With, "Chip-set for video display of multimedia information," *IEEE Transactions on Consumer Electronics*, vol. 45, pp. 706–715, aug 1999.

[29] J. K. Reissmann, S. Bauer, J. Vollmer, B. Schmale, P. Kuhn, J. Kneip, and M. Reissmann, "The MPEG-4 video coding standard-a VLSI point of view," in *Signal Processing Systems*, pp. 43–52, oct 1998.

[30] B. Meyer, *Cost-effective Lifetime and Yield Optimization for NoC-based MPSoCs*. PhD thesis, 2009.

[31] K. Jamieson, M. Malloy, and R. Nowak, "lil ' UCB : An Optimal Exploration Algorithm for Multi-Armed Bandits," *JMLR*, vol. 35, no. 1964, pp. 1–16, 2014.

[32] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. New York: Springer, 2003.

[33] T. Okabe and B. Sendhoff, "A critical survey of performance indices for multi-objective optimisation," *The 2003 Congress on Evolutionary Computation*, vol. 2, pp. 878–885, 2003.