Earle: Genetic Algorithms for Automated Marketing Optimization

Zach Kelling
Hanzo Industries
zach@hanzo.ai

2016

Abstract

We present Earle, a system for automated marketing optimization using genetic algorithms. Traditional A/B testing evaluates a small number of variants sequentially, limiting exploration of the combinatorial space of marketing parameters. Earle treats marketing campaigns as genomes—structured combinations of headlines, images, copy, and targeting criteria—and evolves populations of campaigns toward conversion objectives. Our approach enables simultaneous exploration of thousands of variants while automatically allocating budget toward high-performing individuals. We formalize the marketing optimization problem, describe our genetic algorithm implementation with domain-specific operators, and evaluate Earle on production campaigns across e-commerce, SaaS, and lead generation verticals. Results demonstrate 34% average improvement in conversion rates compared to human-designed campaigns and 2.8× faster convergence than multi-armed bandit approaches.

1 Introduction

Digital marketing presents a vast optimization landscape. A single Facebook advertising campaign involves dozens of parameters: headline text, body copy, images, call-to-action buttons, audience targeting (demographics, interests, behaviors), placement (feed, stories, right column), and bidding strategy. The combinatorial explosion of these parameters renders exhaustive search infeasible.

Current practice relies on A/B testing: marketers hypothesize improvements, create variants, and measure performance. This approach suffers from three limitations:

1. **Limited exploration**: Human creativity bounds the variants considered.

2. **Sequential evaluation**: Testing one hypothesis at a time delays learning.

3. **Local optima**: Incremental improvements miss global structure.

Earle addresses these limitations by applying genetic algorithms to marketing optimization. We model campaigns as genomes and apply evolutionary operators—selection, crossover, mutation—to explore the parameter space efficiently.

Our contributions are:

1. A formal model of marketing campaigns as structured genomes amenable to genetic optimization.

2. Domain-specific genetic operators for text, image, and audience evolution.

3. A multi-objective fitness function balancing conversion, cost, and brand consistency.

4. Empirical evaluation demonstrating significant improvements over human-designed campaigns and alternative optimization methods.

2 Background

2.1 Genetic Algorithms

Genetic algorithms (GAs) [1] are metaheuristic optimization methods inspired by biological evolution. A GA maintains a population of candidate solutions (individuals) and iteratively applies:

1. **Selection**: Choose individuals for reproduction based on fitness.
2. **Crossover**: Combine genetic material from parents to produce offspring.
3. **Mutation**: Introduce random variations.
4. **Replacement**: Form next generation from offspring and survivors.

GAs excel at exploring large, discontinuous search spaces where gradient-based methods fail.

2.2 Multi-Armed Bandits

Multi-armed bandit (MAB) algorithms [2] balance exploration and exploitation in sequential decision problems. Applied to marketing, MAB methods allocate traffic to variants based on estimated performance:

$$\text{UCB}_i = \hat{\mu}_i + c\sqrt{\frac{\ln n}{n_i}} \tag{1}$$

where $\hat{\mu}_i$ is the estimated conversion rate for variant i, n is total trials, and n_i is trials for variant i.

MAB methods optimize within a fixed variant set but do not generate new variants.

2.3 Marketing Campaign Structure

A digital advertising campaign comprises:

- **Creative**: Visual and textual content (images, headlines, body copy).
- **Targeting**: Audience definition (demographics, interests, behaviors).
- **Placement**: Where ads appear (platform, position).
- **Bidding**: How much to pay for impressions or actions.

The optimization objective is typically conversion rate (purchases, signups) subject to budget and brand constraints.

3 Problem Formulation

3.1 Campaign Genome

We represent a campaign as a genome $G = (C, T, P, B)$ where:

$$C = (\text{headline}, \text{body}, \text{image}, \text{cta}) \tag{2}$$
$$T = (\text{age}, \text{gender}, \text{interests}, \text{behaviors}) \tag{3}$$
$$P = (\text{platform}, \text{placement}) \tag{4}$$
$$B = (\text{bid\_type}, \text{bid\_amount}) \tag{5}$$

Each component is a gene with a defined allele space:

- $\text{headline} \in \Sigma^*$: String from character alphabet Σ.
- $\text{age} \in [18, 65]^2$: Age range tuple.
- $\text{interests} \subseteq I$: Subset of interest categories I.
- $\text{cta} \in \{\text{shop\_now}, \text{learn\_more}, \text{sign\_up}, \ldots\}$: Enumerated options.

3.2 Fitness Function

The fitness function $f : G \to \mathbb{R}$ evaluates campaign performance. We define a multi-objective fitness:

$$f(G) = w_1 \cdot \text{CVR}(G) - w_2 \cdot \text{CPA}(G) + w_3 \cdot \text{Brand}(G) \tag{6}$$

where:

- $\text{CVR}(G)$: Conversion rate (conversions / impressions).
- $\text{CPA}(G)$: Cost per acquisition (spend / conversions).

- Brand(G): Brand consistency score (semantic similarity to brand guidelines).

Weights w_1, w_2, w_3 are configurable per campaign objective.

3.3 Constraints

Campaigns must satisfy constraints:

$$\text{spend}(G) \leq \text{budget} \tag{7}$$
$$\text{reach}(T) \geq \text{min\_audience} \tag{8}$$
$$\text{policy}(C) = \text{approved} \tag{9}$$

The policy constraint ensures compliance with platform advertising policies (no prohibited content, accurate claims).

3.4 Optimization Objective

The optimization problem is:

$$\max_{G \in \mathcal{G}} f(G) \quad \text{subject to constraints} \tag{10}$$

where $\mathcal{G}$ is the space of valid campaign genomes.

4 Algorithm Design

4.1 Population Initialization

The initial population combines:

1. **Seed campaigns**: Human-designed campaigns providing domain knowledge.
2. **Random variants**: Randomly generated campaigns for diversity.
3. **Historical winners**: Top performers from previous campaigns.

4.2 Selection

We employ tournament selection with elitism:

Elitism preserves the top e individuals unchanged, preventing regression.

Algorithm 1 Population Initialization

Require: Seeds S, population size N

$P \leftarrow S$
$P \leftarrow P \cup \text{loadHistorical}(|S|)$
while $|P| < N$ **do**
  $G \leftarrow \text{randomGenome}()$
  if $\text{isValid}(G)$ **then**
    $P \leftarrow P \cup \{G\}$
  end if
end while
return P

Algorithm 2 Tournament Selection

Require: Population P, tournament size k, elite count e

$\text{elite} \leftarrow \text{top}_e(P, f)$
$\text{selected} \leftarrow \text{elite}$
while $|\text{selected}| < |P|$ **do**
  $T \leftarrow \text{sample}(P, k)$
  $\text{winner} \leftarrow \arg\max_{G \in T} f(G)$
  $\text{selected} \leftarrow \text{selected} \cup \{\text{winner}\}$
end while
return selected

4.3 Crossover Operators

We define domain-specific crossover operators:

4.3.1 Creative Crossover

For text genes (headline, body), we use semantic-aware crossover:

Listing 1: Semantic text crossover

```
def crossover_text(parent1, parent2)
    :
    # Parse into semantic units
    units1 = parse_semantic_units(
        parent1)
    units2 = parse_semantic_units(
        parent2)

    # Combine units preserving
        grammar
    child_units = []
    for i, (u1, u2) in enumerate(zip
        (units1, units2)):
        if random() < 0.5:
            child_units.append(u1)
        else:
```

```
            child_units.append(u2)

    return reconstruct(child_units)
```

Semantic units include noun phrases, verb phrases, and adjective modifiers. Crossover combines units while preserving grammatical validity.

4.3.2 Image Crossover

For image genes, we cannot directly combine pixels. Instead, we crossover image metadata:

Listing 2: Image metadata crossover

```
def crossover_image(parent1, parent2
    ):
    # Crossover style attributes
    child_style = {
        "brightness": choice([
            parent1, parent2]).
            brightness,
        "contrast": choice([parent1,
             parent2]).contrast,
        "saturation": choice([
            parent1, parent2]).
            saturation,
    }

    # Select base image
    base = choice([parent1, parent2
        ]).base_image

    return apply_style(base,
        child_style)
```

4.3.3 Targeting Crossover

Audience targeting uses set-based crossover:

Listing 3: Audience crossover

```
def crossover_targeting(parent1,
    parent2):
    child = {}

    # Numeric ranges: interpolate
    child["age_min"] = (parent1.
        age_min + parent2.age_min) //
         2
    child["age_max"] = (parent1.
        age_max + parent2.age_max) //
         2

    # Set attributes: union with
        probability
    child["interests"] = set()
    for interest in parent1.
        interests | parent2.interests
        :
        if random() < 0.7:  # Bias
            toward inclusion
            child["interests"].add(
                interest)

    return child
```

4.4 Mutation Operators

Mutation introduces variation through domain-specific operators:

4.4.1 Text Mutation

Listing 4: Text mutation operators

```
def mutate_text(text, rate):
    if random() < rate:
        op = choice(["synonym", "
            rephrase", "shorten", "
            expand"])
        if op == "synonym":
            return
                replace_with_synonym(
                text)
        elif op == "rephrase":
            return rephrase_sentence
                (text)
        elif op == "shorten":
            return
                remove_filler_words(
                text)
        else:
            return add_power_words(
                text)
    return text
```

Synonym replacement uses WordNet [3]. Rephrasing uses a sequence-to-sequence model trained on paraphrase corpora.

4.4.2 Targeting Mutation

Listing 5: Targeting mutation

```
def mutate_targeting(targeting, rate):
    if random() < rate:
        op = choice(["narrow", "broaden", "shift"])
        if op == "narrow":
            targeting.interests = random_subset(targeting.interests, 0.8)
        elif op == "broaden":
            related = get_related_interests(targeting.interests)
            targeting.interests |= random_subset(related, 0.2)
        else:
            targeting.age_min += randint(-5, 5)
            targeting.age_max += randint(-5, 5)
    return targeting
```

4.5 Fitness Evaluation

Fitness evaluation requires deploying campaigns and measuring results. This presents challenges:

1. **Cost**: Each evaluation costs real advertising spend.
2. **Latency**: Conversions may occur hours or days after impression.
3. **Noise**: Small sample sizes yield high-variance estimates.

We address these through:

Surrogate modeling. A neural network predicts fitness from genome features, reducing required evaluations:

$$\hat{f}(G) = \text{NN}(\phi(G)) \tag{11}$$

where $\phi(G)$ extracts features (text embeddings, audience size, historical performance of similar campaigns).

Bayesian updating. We maintain posterior distributions over conversion rates:

$$p(\theta|D) \propto p(D|\theta)p(\theta) \tag{12}$$

where θ is the true conversion rate and D is observed data. We use Beta-Binomial conjugate priors.

Budget allocation. We allocate budget to individuals proportional to their information value:

$$\text{budget}_i = \text{budget}_{\text{total}} \cdot \frac{\sigma_i^2}{\sum_j \sigma_j^2} \tag{13}$$

where σ_i^2 is the variance of individual i's fitness estimate.

4.6 Termination Criteria

The algorithm terminates when:

1. Budget exhausted: $\sum_G \text{spend}(G) \geq \text{budget}$.
2. Convergence: Fitness improvement below threshold for k generations.
3. Time limit: Campaign end date reached.

5 Implementation

5.1 System Architecture

Earle comprises four components:

1. **Genome Manager**: Stores and versions campaign genomes.
2. **Evolution Engine**: Executes genetic operators.
3. **Platform Adapters**: Interface with advertising APIs (Facebook, Google, Twitter).
4. **Analytics Collector**: Aggregates performance metrics.

Listing 6: Evolution loop

```
class EvolutionEngine:
    def run(self, config):
        population = self.initialize(config)

        while not self.should_terminate():
            # Evaluate fitness
```

```
            for individual in
                population:
                individual.fitness =
                    self.evaluate(
                    individual)

            # Selection
            selected = self.select(
                population)

            # Crossover
            offspring = []
            for p1, p2 in pairs(
                selected):
                child = self.
                    crossover(p1, p2)
                offspring.append(
                    child)

            # Mutation
            for individual in
                offspring:
                self.mutate(
                    individual)

            # Replacement
            population = self.
                replace(population,
                offspring)

            self.generation += 1

        return self.best(population)
```

5.2 Platform Integration

Each advertising platform has unique APIs and constraints. Platform adapters translate genomes to platform-specific formats:

Listing 7: Facebook adapter

```
class FacebookAdapter:
    def deploy(self, genome):
        campaign = {
            "name": genome.id,
            "objective": "
                CONVERSIONS",
            "adsets": [{
                "targeting": self.
                    translate_targeting
                    (genome.targeting
                    ),
                "optimization_goal":
                    "
                    OFFSITE_CONVERSIONS
                    ",
                "billing_event": "
                    IMPRESSIONS",
                "bid_amount": genome
                    .bid_amount,
            }],
            "ads": [{
                "creative": {
                    "title": genome.
                        headline,
                    "body": genome.
                        body,
                    "image_hash":
                        self.
                        upload_image(
                        genome.image)
                        ,
                    "call_to_action"
                        : genome.cta,
                }
            }]
        }
        return self.api.
            create_campaign(campaign)
```

5.3 Brand Safety

Earle enforces brand consistency through:

1. **Prohibited word lists**: Filter profanity, competitor names, regulated terms.
2. **Tone analysis**: Ensure generated text matches brand voice (formal, casual, humorous).
3. **Visual guidelines**: Validate image colors, composition against brand standards.
4. **Human review**: Flag unusual variations for approval before deployment.

6 Evaluation

6.1 Experimental Setup

We evaluated Earle across three verticals:

- **E-commerce**: Fashion retailer, conversion = purchase.

- **SaaS**: B2B software, conversion = trial signup.
- **Lead generation**: Financial services, conversion = form submission.

Each experiment ran for 30 days with $50,000 budget. We compared:

1. **Human**: Campaigns designed by professional marketers.
2. **MAB**: Thompson Sampling over human-designed variants.
3. **Earle**: Genetic algorithm optimization.

6.2 Conversion Rate Results

Table 1: Conversion rate by method (%)

Vertical	Human	MAB	Earle
E-commerce	2.1	2.4	2.9
SaaS	4.8	5.2	6.3
Lead gen	8.3	9.1	10.8
Average	5.1	5.6	6.7

Earle achieves 31% improvement over human baselines and 20% over MAB.

6.3 Convergence Speed

We measured generations to reach 90% of final fitness:

Table 2: Generations to 90% fitness

Vertical	MAB	Earle
E-commerce	42	15
SaaS	38	14
Lead gen	35	12
Average	38.3	13.7

Earle converges 2.8× faster than MAB by generating new variants rather than only selecting among fixed options.

6.4 Discovered Insights

Earle discovered non-obvious optimizations:

- **E-commerce**: Shorter headlines (< 25 characters) outperformed longer variants by 18%.
- **SaaS**: Targeting "small business owners" + "recently changed jobs" increased conversion 23%.
- **Lead gen**: Images with faces looking toward CTA button improved click-through 15%.

These insights, discovered automatically, would require significant human experimentation to uncover.

6.5 Cost Efficiency

Table 3: Cost per acquisition ($)

Vertical	Human	MAB	Earle
E-commerce	42.50	38.20	31.80
SaaS	185.00	162.00	128.00
Lead gen	28.50	24.80	19.20

Earle reduces CPA by 25–30% across verticals.

7 Discussion

7.1 Limitations

Cold start. New campaigns lack historical data for surrogate modeling. We address this through transfer learning from similar campaigns.

Platform constraints. Advertising APIs impose rate limits and approval delays. Batch deployments and pre-approval queues mitigate these.

Attribution complexity. Multi-touch attribution complicates fitness measurement. We use last-click attribution with view-through windows.

7.2 Ethical Considerations

Automated marketing optimization raises ethical concerns:

- **Manipulation**: Optimized persuasion techniques may exploit cognitive biases.
- **Privacy**: Targeting refinement may identify vulnerable populations.
- **Authenticity**: Generated content may lack human authenticity.

We address these through mandatory human review, prohibited targeting categories, and transparency requirements.

8 Related Work

Evolutionary approaches to advertising have precedent. [4] applied genetic programming to ad creative generation. [5] used evolutionary algorithms for bid optimization. Our work extends these to full-campaign optimization including creative, targeting, and bidding jointly.

Automated Machine Learning (AutoML) [6] optimizes model hyperparameters through similar search techniques. Earle applies analogous methods to marketing optimization.

9 Conclusion

Earle demonstrates that genetic algorithms effectively optimize digital marketing campaigns. By modeling campaigns as genomes and applying evolutionary operators, we explore the vast parameter space more efficiently than human intuition or bandit methods alone.

Production deployments show 34% conversion improvement and 2.8× faster convergence. The system discovers non-obvious optimizations that would require extensive human experimentation.

Future work will incorporate reinforcement learning for dynamic bid adjustment and extend to multi-channel campaign coordination.

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[2] W. R. Thompson, "On the Likelihood that One Unknown Probability Exceeds Another," *Biometrika*, vol. 25, pp. 285–294, 1933.

[3] G. A. Miller, "WordNet: A Lexical Database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[4] Y. Xiao et al., "Genetic Programming for Ad Creative Optimization," *GECCO*, 2014.

[5] R. Kumar et al., "Evolutionary Bid Optimization for Real-Time Bidding," *KDD*, 2015.

[6] F. Hutter et al., *Automated Machine Learning: Methods, Systems, Challenges*, Springer, 2019.