On Constrained Boolean Pareto Optimization*

Chao Qian and **Yang Yu** and **Zhi-Hua Zhou**
National Key Laboratory for Novel Software Technology, Nanjing University
Collaborative Innovation Center of Novel Software Technology and Industrialization
Nanjing 210023, China
{qianc,yuy,zhouzh}@lamda.nju.edu.cn

Abstract

Pareto optimization solves a constrained optimization task by reformulating the task as a bi-objective problem. Pareto optimization has been shown quite effective in applications; however, it has little theoretical support. This work theoretically compares Pareto optimization with a penalty approach, which is a common method transforming a constrained optimization into an unconstrained optimization. We prove that on two large classes of constrained Boolean optimization problems, minimum matroid optimization (P-solvable) and minimum cost coverage (NP-hard), Pareto optimization is more efficient than the penalty function method for obtaining the optimal and approximate solutions, respectively. Furthermore, on a minimum cost coverage instance, we also show the advantage of Pareto optimization over a greedy algorithm.

1 Introduction

Optimization tasks usually come with constraints [Bertsekas, 1999], which must be satisfied by the final solutions. For general constrained optimization, i.e., we don't make assumptions on its objective function and constraints, the most widely used method is the *penalty function method* [Ben Hadj-Alouane and Bean, 1997; Coello Coello, 2002]. This method forms an unconstrained optimization problem by adding a penalty term to the objective function of the original problem, where the penalty term usually measures the degree of the constraints violation. When the penalty term is well balanced with the objective function, the unconstrained optimization problem will yield the same optimal solutions as the original constrained optimization problem.

Pareto optimization [Coello Coello, 2002; Cai and Wang, 2006] solves a constrained optimization problem in a new way: it treats the constraints violation degree as an additional objective, and reformulates the original constrained optimization problem as a bi-objective optimization problem, which is to optimize the original objective function and to minimize the violation degree simultaneously; the bi-objective optimization problem is then solved by a multi-objective evolutionary algorithm; and finally the produced solutions are transformed back to the original problem. Previously, the performance of this approach was only tested in experiments on some benchmark problems [Venkatraman and Yen, 2005; Cai and Wang, 2006]. Recently it has been applied to the ensemble pruning problem [Qian *et al.*, 2015], where Pareto optimization is shown to be superior to state-of-the-art methods both theoretically and empirically [Qian *et al.*, 2015]. However, the theoretical understanding of Pareto optimization is still quite insufficient. It is also important to theoretically understand how widely Pareto optimization is applicable.

This paper tries to shed some light on the power of Pareto optimization. We theoretically compare the efficiency of Pareto optimization with a penalty function method on two large Boolean optimization problem classes. The efficiency is measured by the expected running time for achieving an optimal or approximate solution. Given a problem class $\mathcal{F}$, we compare the worst case running time [Cormen *et al.*, 2001]: denoting $POM(f)$ and $PFM(f)$ as the expected running time of the Pareto optimization and the penalty function method for solving $f \in \mathcal{F}$, respectively, we compare $POM(\mathcal{F}) = \max\{POM(f) | f \in \mathcal{F}\}$ with $PFM(\mathcal{F}) = \max\{PFM(f) | f \in \mathcal{F}\}$. Our main results are:

- On the P-solvable minimum matroid optimization problem (denoted as $\mathcal{M}$) for finding an optimal solution, $POM(\mathcal{M}) \in O(rn(\log n + \log w_{\max} + r))$ (Theorem 1) and $PFM(\mathcal{M}) \in \Omega(r^2 n(\log n + \log w_{\max}))$ (Theorem 2), where n, $w_{\max}$ and r are parameters of problem size, maximum weight and matroid ranking, respectively. Thus, Pareto optimization is faster than the penalty function method by at least a factor of $\min\{\log n + \log w_{\max}, r\}$.
- On the NP-hard minimum cost coverage problem (denoted as $\mathcal{C}$) for finding an approximate solution, $POM(\mathcal{C}) \in O(Nn(\log n + \log w_{\max} + N))$ (Theorem 3) and $PFM(\mathcal{C})$ is at least exponential w.r.t. n, N and $\log w_{\max}$ (Theorem 4), where N is a submodular function parameter. Thus, Pareto optimization is exponentially faster than the penalty function method.
- On a specific minimum cost coverage instance $C \in \mathcal{C}$, the greedy algorithm finds a local optimal solution (Proposition 1), while $POM(C) \in O(n^2 \log n)$ (Proposition 2) and $PFM(C) \in O(n^3 \log n)$ (Proposition 3) for finding

*This research was supported by NSFC (61375061, 61333014) and JiangsuSF (BK2012303).

the global optimal solution. Thus, Pareto optimization can be better than the greedy algorithm.

The rest of the paper starts with a section of preliminaries. Section 3 and 4 present the studies on minimum matroid optimization and minimum cost coverage, respectively. We further compare Pareto optimization with a greedy algorithm in section 5. Section 6 concludes the paper.

2 Preliminaries

2.1 Constrained Optimization

Constrained optimization [Bertsekas, 1999] is to optimize an objective function with constraints that must be satisfied. It can be formally stated as follows.

Definition 1 (Constrained Optimization)

$$\begin{aligned} \arg\min_{\boldsymbol{x}\in\{0,1\}^n} \quad & f(\boldsymbol{x}) & & \quad (1)\\ \textit{subject to} \quad & g_i(\boldsymbol{x}) = 0 & \textit{for } 1 \le i \le q, & \\ & h_i(\boldsymbol{x}) \le 0 & \textit{for } q+1 \le i \le m, & \end{aligned}$$

where $f(\boldsymbol{x})$ is the objective function, $g_i(\boldsymbol{x})$ and $h_i(\boldsymbol{x})$ are the equality and inequality constraints, respectively.

Note that we consider binary spaces $\{0,1\}^n$ in this paper. We also only consider minimization since maximizing f is equivalent to minimizing $-f$. A solution is (in)feasible if it does (not) satisfy the constraints. Thus, constrained optimization is to find a feasible solution minimizing the objective f.

2.2 The Penalty Function Method

The penalty function method [Ben Hadj-Alouane and Bean, 1997; Coello Coello, 2002] is the most widely employed method to solve constrained optimization. For the optimization problem in Eq. (1), the penalty function method turns to solve an unconstrained optimization problem:

$$\arg\min_{\boldsymbol{x}\in\{0,1\}^n} \quad f(\boldsymbol{x}) + \lambda \sum\nolimits_{i=1}^{m} f_i(\boldsymbol{x}),$$

where f is the objective in Eq. (1), λ is the penalty coefficient, and f_i is the penalty term expressing a violation degree of the i-th constraint. A straightforward way to set f_i is

$$f_i(\boldsymbol{x}) = \begin{cases} |g_i(\boldsymbol{x})| & 1 \le i \le q, \\ \max\{0, h_i(\boldsymbol{x})\} & q+1 \le i \le m. \end{cases} \quad (2)$$

For the penalty coefficient λ, a static value has been shown to be simple and robust [Homaifar *et al.*, 1994; Michalewicz and Schoenauer, 1996]. Furthermore, Zhou and He [2007] proved that a large enough static λ value makes the penalty function method equivalent to the "superiority of feasible points" strategy [Deb, 2000] that first prefers a smaller constraints violation degree, and then compares the objective value if having the same violation degree. Such a strategy ensures that a feasible solution is always better than an infeasible one, and thus the optimal solutions must be feasible and are the same as that of the constrained optimization problem.

Then, an unconstrained optimization algorithm will be employed to solve this unconstrained problem. For tackling various constrained optimization problems, we consider the context of general purpose optimization algorithms, such as randomized local search and evolutionary algorithms [Bäck, 1996]. The penalty function method with a general purpose optimization algorithm [He and Yao, 2001; Yu and Zhou, 2008] is presented in Algorithm 1.

Algorithm 1 (The Penalty Function Method) *Given a constrained optimization problem as in Eq.* (1)*, it contains:*

```
1. Let $h(x) = f(x) + \lambda\sum_{i=1}^{m} f_i(x)$ according to Eq. (2).
2. $x$ = selected from $\{0,1\}^n$ uniformly at random.
3. repeat until the termination condition is met
4.     $x'$ = flip each bit of $x$ independently with prob. $\frac{1}{n}$.
5.     if $h(x') \le h(x)$
6.         $x = x'$.
7. return $x$
```

Sharing a common structure with other general purpose optimization algorithms, it starts from a random solution, generates a new solution from the current solution by some heuristic, decides if the new solution is good enough to be kept, and repeats the generation and decision until a good solution is found. It employs a typical heuristic for generating new solutions in evolutionary algorithms, that flips each bit (0 to 1 or inverse) of a solution with probability $\frac{1}{n}$. This algorithm does not make any assumption on the optimization problem, but only require that two solutions can be compared for the goodness (i.e., step 5). Thus it can be widely applied.

2.3 The Pareto Optimization Method

By Pareto optimization [Coello Coello, 2002; Cai and Wang, 2006], the constrained optimization problem is reformulated into a multi-objective (bi-objective) problem:

$$\arg\min_{\boldsymbol{x}\in\{0,1\}^n} \left(f(\boldsymbol{x}), \sum\nolimits_{i=1}^{m} f_i(\boldsymbol{x})\right)$$

where f_i is the constraint violation degree and can be set as Eq. (2). In this bi-objective formulation, a solution has an objective vector instead of a scalar objective value. Unlike single-objective optimization, the objective vector makes the comparison between two solutions not straightforward, because it is possible that one solution is better on the first dimension while the other is better on the second dimension. For this situation, the domination relationship between solutions is usually used, which is introduced in Definition 2 for the bi-objective (i.e., two objectives) minimization case.

Definition 2 (Domination) *Let $\boldsymbol{g} = (g_1, g_2) : \mathcal{X} \to \mathbb{R}^2$ be the objective vector. For two solutions $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$:*
(1) $\boldsymbol{x}$ weakly dominates *$\boldsymbol{x}'$ if $g_1(\boldsymbol{x}) \le g_1(\boldsymbol{x}')$ and $g_2(\boldsymbol{x}) \le g_2(\boldsymbol{x}')$, denoted as $\succeq_{\boldsymbol{g}}$;*
(2) $\boldsymbol{x}$ dominates *$\boldsymbol{x}'$ if $\boldsymbol{x} \succeq_{\boldsymbol{g}} \boldsymbol{x}'$ and either $g_1(\boldsymbol{x}) < g_1(\boldsymbol{x}')$ or $g_2(\boldsymbol{x}) < g_2(\boldsymbol{x}')$, denoted as $\succ_{\boldsymbol{g}}$.*

Thus, a bi-objective optimization problem may not have a single optimal solution, but instead have a set of *Pareto optimal* solutions. A solution $\boldsymbol{x}$ is Pareto optimal if there is no other solution in $\mathcal{X}$ that dominates $\boldsymbol{x}$. The set of objective vectors of all the Pareto optimal solutions is called the *Pareto front*.

For the fairness of comparison, a general purpose bi-objective optimization algorithm [Laumanns *et al.*, 2004;

Qian *et al.*, 2013] with the same initialization and random perturbation as Algorithm 1 is then employed to solve this bi-objective problem, presented in Algorithm 2.

Algorithm 2 (The Pareto Optimization Method) *Given a constrained optimization problem as in Eq.* (1)*, it contains:*

```
1. Let $\boldsymbol{g}(\boldsymbol{x}) = (f(\boldsymbol{x}), \sum_{i=1}^{m} f_i(\boldsymbol{x}))$.
2. $\boldsymbol{x}$ = selected from $\{0,1\}^n$ uniformly at random.
3. $P = \{\boldsymbol{x}\}$.
4. repeat until the termination condition is met
5.    $\boldsymbol{x}$ = selected from $P$ uniformly at random.
6.    $\boldsymbol{x'}$ = flip each bit of $\boldsymbol{x}$ independently with prob. $\frac{1}{n}$.
7.    if $\nexists \boldsymbol{z} \in P$ such that $\boldsymbol{z} \succ_{\boldsymbol{g}} \boldsymbol{x'}$
8.       $P = (P - \{\boldsymbol{z} \in P | \boldsymbol{x'} \succeq_{\boldsymbol{g}} \boldsymbol{z}\}) \cup \{\boldsymbol{x'}\}$.
9. return $\boldsymbol{x} \in P$ with $\sum_{i=1}^{m} f_i(\boldsymbol{x}) = 0$
```

It firstly generates a random solution, and puts it into the candidate solution set P; and then follows a cycle to improve the solutions in P iteratively. In each iteration, a solution $\boldsymbol{x}$ is randomly selected from P; and is then perturbed to generate a new solution $\boldsymbol{x'}$; if $\boldsymbol{x'}$ is not dominated by any solution in P, $\boldsymbol{x'}$ is added into P and at the same time solutions in P that are weakly dominated by $\boldsymbol{x'}$ get removed. Finally, the solution in P that violates no constraints is selected.

Note that Pareto optimization here is different from traditional multi-objective optimization [Deb *et al.*, 2002]. The latter aims at finding a uniform approximation of the Pareto front, while Pareto optimization aims at finding the optimal solutions for the original constrained optimization problem.

2.4 Running Time Analysis Comparison

To measure the performance of an algorithm solving a problem, we employ the expected running time complexity, which counts the number of objective function evaluations (the most time-consuming step) until finding a desired solution, since it has been a leading theoretical aspect for randomized search heuristics [Neumann and Witt, 2010; Auger and Doerr, 2011]. Because both the two analyzed methods perform only one objective evaluation (i.e., $h(\boldsymbol{x'})$ and $\boldsymbol{g}(\boldsymbol{x'})$) in each iteration, the expected running time equals the average number of iterations. For P-solvable problems, we use exact analysis [He and Yao, 2001], i.e., the running time until an optimal solution is found. For NP-hard problems, we use approximate analysis [Yu *et al.*, 2012], i.e., the running time until finding an α-approximate solution having the objective value $\leq \alpha \cdot OPT$, where $\alpha > 1$ and OPT is the optimal function value.

3 On Minimum Matroid Optimization

We first analyze the P-solvable minimum matroid optimization problem [Edmonds, 1971], which covers several combinatorial optimization problems, e.g., minimum spanning tree.

Let $|\cdot|$ denote the size (i.e., cardinality) of a set. A matroid is a pair (U, S), where U is a finite set and $S \subseteq 2^U$, satisfying

(1) $\emptyset \in S$; (2) $\forall A \subseteq B \in S, A \in S$;
(3) $\forall A, B \in S, |A| > |B| : \exists e \in A - B, B \cup \{e\} \in S$.

The elements of S are called *independent*. For any $A \subseteq U$, a maximal independent subset of A is called a basis of A; the rank of A is the maximal cardinality of a basis of A, i.e., $r(A) = \max\{|B| \mid B \subseteq A, B \in S\}$. Note that for a matroid, all bases of A have the same cardinality.

Given a matroid (U, S) and a weight function $w : U \to \mathbb{N}$ where $\mathbb{N}$ denotes the positive integer set, the minimum matroid optimization problem is to find a basis of U with the minimum weight. Let $U = \{e_1, e_2, \ldots, e_n\}$ and $w_i = w(e_i)$. Let $\boldsymbol{x} \in \{0,1\}^n$ represent a subset of U, where $x_i = 1$ means that e_i is selected. For notational convenience, we will not distinguish $\boldsymbol{x}$ and its corresponding subset $\{e_i \in U | x_i = 1\}$. The problem then can be formally stated as follows.

Definition 3 (Minimum Matroid Optimization) *Given a matroid (U, S) and a weight function $w : U \to \mathbb{N}$, it is to find a solution such that*

$$\arg\min_{\boldsymbol{x} \in \{0,1\}^n} \; w(\boldsymbol{x}) = \sum\nolimits_{i=1}^{n} w_i x_i \quad s.t. \quad r(\boldsymbol{x}) = r(U). \quad (3)$$

Note that, a rank oracle that computes the rank of a subset is polynomially equivalent to an independence oracle that tells whether a subset is independent [Korte and Vygen, 2012].

For Pareto optimization, noting $r(U) \geq r(\boldsymbol{x})$, the objective vector $\boldsymbol{g}$ is implemented as

$$\boldsymbol{g}(\boldsymbol{x}) = \big(w(\boldsymbol{x}), r(U) - r(\boldsymbol{x})\big).$$

Theorem 1 proves the running time bound, where $w_{\max} = \max\{w_i \mid 1 \leq i \leq n\}$ is the maximum element weight in U.

Theorem 1 *For minimum matroid optimization, the expected running time of the Pareto optimization method for finding an optimal solution is $O(r(U)n(\log n + \log w_{\max} + r(U)))$.*

The proof idea is to divide the optimization process into two phases: (1) starts after initialization and finishes until finding the special solution $\{0\}^n$; (2) starts from $\{0\}^n$ and follows the greedy behavior [Edmonds, 1971] to find an optimal solution. The running time of phase (1) shown in Lemma 2 is derived by applying multiplicative drift analysis (i.e., Lemma 1), a recently proposed approach for analyzing the hitting time of a random process.

Lemma 1 (Multiplicative Drift Analysis) *[Doerr* et al.*, 2012] Let $S \subseteq \mathbb{R}^+$ be a finite set of positive numbers with minimum $s_{\min}$. Let $\{X_t\}_{t \in \mathbb{N}}$ be a sequence of random variables over $S \cup \{0\}$. Let T be the random variable that denotes the first point in time $t \in \mathbb{N}$ for which $X_t = 0$. Suppose that there exists a real number $\delta > 0$ such that*

$$\mathbb{E}[\![X_t - X_{t+1} | X_t = s]\!] \geq \delta s$$

holds for all $s \in S$. Then for all $s_0 \in S$, we have

$$\mathbb{E}[\![T | X_0 = s_0]\!] \leq (1 + \log(s_0 / s_{min})) / \delta.$$

Lemma 2 *For minimum matroid optimization, the expected running time of the Pareto optimization method for finding $\{0\}^n$ is $O(r(U)n(\log n + \log w_{\max}))$.*

Proof. We prove it by using Lemma 1. Let $X_t = \min\{w(\boldsymbol{x}) \mid \boldsymbol{x} \in P\}$ (i.e., the minimum weight of solutions in the candidate solution set P) after t iterations of Algorithm 2. $X_t = 0$ implies that the solution $\{0\}^n$ has been found, and thus T is the running time for finding $\{0\}^n$.

Then, we are to analyze $\mathbb{E}[\![X_t - X_{t+1} | X_t]\!]$. Let $\boldsymbol{x}$ be the corresponding solution with $w(\boldsymbol{x}) = X_t$. It is easy to see that X_t cannot increase (i.e., $X_{t+1} \leq X_t$) because a newly generated solution $\boldsymbol{x'}$ with a larger weight cannot weakly dominate $\boldsymbol{x}$. Let $P_{\max}$ denote the largest size of P during the optimization. In the $(t+1)$-th iteration, we consider that $\boldsymbol{x}$ is selected in step 5 of Algorithm 2, which happens with probability at least $\frac{1}{P_{\max}}$. Then, in step 6, we denote $\boldsymbol{x^i}$ as the solution generated by flipping the i-th bit (i.e., x_i) of $\boldsymbol{x}$ and keeping other bits unchanged, which happens with probability $\frac{1}{n}(1-\frac{1}{n})^{n-1} \geq \frac{1}{en}$. If $x_i = 1$, the generated solution $\boldsymbol{x^i}$ will be included into P since it has the smallest weight now and X_{t+1} will decrease to $X_t - w_i$. Thus, we have

$$\mathbb{E}[\![X_{t+1}]\!] \leq \sum_{i:x_i=1} \frac{X_t - w_i}{enP_{\max}} + (1 - \sum_{i:x_i=1} \frac{1}{enP_{\max}})X_t$$
$$= X_t - w(\boldsymbol{x})/(enP_{\max}) = (1 - 1/(enP_{\max}))X_t.$$

This implies that $\mathbb{E}[\![X_t - X_{t+1} | X_t]\!] \geq X_t/(enP_{\max})$.

The procedure of Algorithm 2 ensures that the solutions maintained in P must be incomparable, which implies that there exists at most one solution in P for each value of one objective. Since the objective $r(U) - r(\boldsymbol{x}) \in \{0, 1, \ldots, r(U)\}$, $P_{\max} \leq r(U) + 1$. Note that $X_0 \leq nw_{\max}$ and $s_{\min} \geq 1$ by $w_i \in \mathbb{N}$. Thus, by Lemma 1, for any X_0,

$$\mathbb{E}[\![T | X_0]\!] \leq en(r(U) + 1)(1 + \log(nw_{\max}))$$
$$\in O(r(U)n(\log n + \log w_{\max})).$$

□

Proof of Theorem 1. We analyze phase (2) after finding $\{0\}^n$. Let $w^k = \min\{w(\boldsymbol{x}) \mid r(\boldsymbol{x}) = k\}$ denote the smallest weight in all the solutions with rank k. It is easy to see that w^k increases with k. Then the Pareto front of $\boldsymbol{g}$ is $\{(w^k, r(U) - k) \mid 0 \leq k \leq r(U)\}$. We are to show that staring from $\{0\}^n$ with $(w^0 = 0, r(U))$, the algorithm can find the solutions corresponding to the Pareto front from $k = 0$ to $r(U)$ sequentially, where the last solution with $(w^{r(U)}, 0)$ is just the the optimal solution of Eq. (3) we are to find.

We first prove that for any $\boldsymbol{x}$ corresponding to $(w^k, r(U) - k)$, adding the lightest element that makes $r(\boldsymbol{x})$ increase by 1 will produce a solution $\boldsymbol{x'}$ corresponding to $(w^{k+1}, r(U) - k - 1)$. Assume that $\boldsymbol{x} = \{e_{i_1}, \ldots, e_{i_k}\}$, $\boldsymbol{x'} = \boldsymbol{x} \cup \{e_{i_{k+1}}\}$, and there exists another solution $\hat{\boldsymbol{x}} = \{e_{j_1}, \ldots, e_{j_k}, e_{j_{k+1}}\}$ with rank $k{+}1$ and $w(\hat{\boldsymbol{x}}) < w(\boldsymbol{x'})$. Assume that $w_{j_1} \leq \ldots \leq w_{j_{k+1}}$. If $w_{j_{k+1}} \geq w_{i_{k+1}}$, $w(\hat{\boldsymbol{x}}) - w_{j_{k+1}} < w(\boldsymbol{x'}) - w_{i_{k+1}} = w(\boldsymbol{x})$, which contradicts with that $\boldsymbol{x}$ has the smallest weight in all the solutions with rank k. If $w_{j_{k+1}} < w_{i_{k+1}}$, then $\exists e \in \hat{\boldsymbol{x}} - \boldsymbol{x}$, $\boldsymbol{x} \cup \{e\}$ has rank $k + 1$ and $w(e) \leq w_{j_{k+1}} < w_{i_{k+1}}$, which contradicts with that $e_{i_{k+1}}$ is the lightest element that increases $r(\boldsymbol{x})$ by 1. Thus, our claim holds.

Assume that $\{(w^k, r(U) - k) \mid 0 \leq k \leq i\}$ has been found. This is well defined because $i \geq 0$ after finding $\{0\}^n$. Based on the above analysis, for finding $(w^{i+1}, r(U) - i - 1)$ in one iteration, it is sufficient to select the solution corresponding to $(w^i, r(U) - i)$ in step 5 of Algorithm 2 and flip only the lightest element that increases its rank by 1 in step 6, which happens with probability at least $\frac{1}{P_{\max}} \cdot \frac{1}{n}(1 - \frac{1}{n})^{n-1} \geq \frac{1}{en(r(U)+1)}$. Then, the expected running time is $O(r(U)n)$ for finding $(w^{i+1}, r(U) - i - 1)$. Since $r(U)$ such processes are sufficient to find the optimal solution with $(w^{r(U)}, 0)$, the expected running time of phase (2) is $O(r^2(U)n)$.

By combining the two phases, the total expected running time is $O(r(U)n(\log n + \log w_{\max} + r(U)))$. □

For the penalty function method, we first show that minimum spanning tree (MST) is an instance of minimum matroid optimization and then give a concrete MST example where the penalty function method needs more running time than Pareto optimization.

Given an undirected connected graph $G = (V, E)$ on m vertices and n edges with weights $w : E \to \mathbb{N}$, the MST problem is to find a connected subgraph with the minimum weight. Let $E = \{e_1, e_2, \ldots, e_n\}$. Let $\boldsymbol{x} \in \{0,1\}^n$ represent a subgraph, where $x_i = 1$ means that the edge e_i is selected; let $c(\boldsymbol{x})$ denote its connected components. The MST problem can be formally stated as follows.

Definition 4 (Minimum Spanning Tree) *Given an undirected connected graph $G = (V, E)$ with a weight function $w : E \to \mathbb{N}$, it is to find a solution such that*

$$\arg\min_{\boldsymbol{x} \in \{0,1\}^n} \; w(\boldsymbol{x}) = \sum\nolimits_{i=1}^{n} w_i x_i \quad s.t. \quad c(\boldsymbol{x}) = 1.$$

Let $U = E$ and $S = \{\text{subgraphs without cycles}\}$. Then, (U, S) is a matroid, $c(\boldsymbol{x}) + r(\boldsymbol{x}) = m$ and $r(U) = m - 1$. It is easy to verify that MST is a minimum matroid optimization instance (i.e., Definition 3) with such a configuration.

Neumann and Wegener [2007] have analyzed the running time of the penalty function method on a specific MST example, as shown in Lemma 3 (i.e., Theorem 6 in their paper). Note that the parameters m and n in [Neumann and Wegener, 2007] are opposite to ours, and we have exchanged them here for consistency. Thus, we have Theorem 2 by $r(U) = m - 1$.

Lemma 3 *[Neumann and Wegener, 2007] The expected running time until the penalty function method finds a minimum spanning tree for an example graph ($n \in \Theta(m^2)$, $w_{\max} = 3m^2$) equals $\Theta(n^2 \log m)$.*

Theorem 2 *There exists a minimum matroid optimization instance, where the expected running time of the penalty function method for finding an optimal solution is $\Theta(r^2(U)n(\log n + \log w_{\max}))$.*

4 On Minimum Cost Coverage

We then analyze the minimum cost coverage problem, which is NP-hard. A representative instance is the submodular set cover problem [Wolsey, 1982], which arises in many applications, e.g., social networks [Kempe *et al.*, 2003] and sensor placement [Krause *et al.*, 2008].

Let $U = \{e_1, e_2, \ldots, e_n\}$ be a finite set. A set function $f : 2^U \to \mathbb{R}$ is monotone and submodular iff $\forall A, B \subseteq U$, $f(A) \leq f(B) + \sum_{e \in A-B}(f(B \cup \{e\}) - f(B))$ [Nemhauser *et al.*, 1978]. Let $\boldsymbol{x} \in \{0,1\}^n$ represent a subset of U. Minimum cost coverage can be formally stated as follows.

Definition 5 (Minimum Cost Coverage) *Given a monotone submodular function $f : 2^U \to \mathbb{R}$, some value $q \leq f(U)$ and a weight function $w : U \to \mathbb{N}$, it is to find a solution such that*

$$\arg\min_{\boldsymbol{x} \in \{0,1\}^n} w(\boldsymbol{x}) = \sum\nolimits_{i=1}^{n} w_i x_i \quad s.t. \quad f(\boldsymbol{x}) \geq q. \tag{4}$$

We consider f to be non-negative. Since the rank of a matroid is a monotone submodular function, minimum matroid optimization is actually a minimum cost coverage instance with $q = r(U)$. We analyze them separately because minimum matroid optimization is P-solvable and minimum cost coverage is NP-hard in general. We use exact and approximate analysis, respectively.

For minimum cost coverage by Pareto optimization, the objective vector $\boldsymbol{g}$ is implemented as

$$\boldsymbol{g}(\boldsymbol{x}) = \big(w(\boldsymbol{x}), \max\{0, q - f(\boldsymbol{x})\}\big).$$

The running time bound is proved in Theorem 3. Let $H_i = \sum_{j=1}^{i} 1/j$ be the i-th harmonic number, and let c denote the minimum real number making $c \cdot f(x)$ for all x and $c \cdot q$ to be integer. Let N denote the number of distinct f values in $[0, q)$. The proof idea is similar to that for Theorem 1 except that it follows the greedy behavior [Wolsey, 1982] to find an approximate solution.

Theorem 3 *For minimum cost coverage, the expected running time of the Pareto optimization method for finding a H_{cq}-approximate solution is $O(Nn(\log n + \log w_{\max} + N))$.*

Proof. Because the objective $\max\{0, q - f(\boldsymbol{x})\}$ has $N+1$ number of distinct values and the solutions in P are incomparable, the largest size of P is not larger than $N + 1$. By using the same proof as Lemma 2 except $P_{\max} \leq N + 1$ here, we can derive that the expected running time for finding $\{0\}^n$ is $O(Nn(\log n + \log w_{\max}))$.

Let $R_k = H_{cq} - H_{c(q-k)}$. Let $\boldsymbol{x}^*$ and OPT denote an optimal solution and the optimal function value of Eq. (4), respectively. Then, $w(\boldsymbol{x}^*) = OPT$. Let $K_{\max}$ denote the maximum value of k such that there exists one solution $\boldsymbol{x}$ in P with $\min\{q, f(\boldsymbol{x})\} = k$ and $w(\boldsymbol{x}) \leq R_k \cdot OPT$. That is, $K_{\max} = \max\{k \mid \exists \boldsymbol{x} \in P, \min\{q, f(\boldsymbol{x})\} = k \wedge w(\boldsymbol{x}) \leq R_k \cdot OPT\}$. Then, we are to analyze the expected running time until $K_{\max} = q$, which implies that it finds a R_q (i.e., H_{cq})-approximate solution.

After finding $\{0\}^n$, $K_{\max} \geq 0$. Assume that currently $K_{\max} = k < q$. Let $\boldsymbol{x}$ denote the corresponding solution with the value k, i.e., $\min\{q, f(\boldsymbol{x})\} = k$ and $w(\boldsymbol{x}) \leq R_k \cdot OPT$. We are first to show that $K_{\max}$ cannot decrease. If $\boldsymbol{x}$ is kept in P, $K_{\max}$ obviously will not decrease. If $\boldsymbol{x}$ is deleted, by step 7 and 8 of Algorithm 2, a newly generated solution $\boldsymbol{x'}$ weakly dominating $\boldsymbol{x}$ (i.e., $\boldsymbol{x'}$ is not worse on both the two objectives) must be included. Note that $\max\{0, q - f(\boldsymbol{x})\} = q - \min\{q, f(\boldsymbol{x})\}$. Thus, $\min\{q, f(\boldsymbol{x'})\} = k' \geq \min\{q, f(\boldsymbol{x})\} = k$ and $w(\boldsymbol{x'}) \leq w(\boldsymbol{x})$. Because R_k increases with k, $w(\boldsymbol{x'}) \leq R_k \cdot OPT \leq R_{k'} \cdot OPT$. Thus, $K_{\max} \geq k'$, i.e., $K_{\max}$ will not decrease.

Then, we are to show that $K_{\max}$ can increase by flipping a specific 0 bit of $\boldsymbol{x}$. Let $f'(\boldsymbol{x}) = \min\{q, f(\boldsymbol{x})\}$. Let $\boldsymbol{x^i}$ denote the solution generated by flipping the i-th bit of $\boldsymbol{x}$. Let $I = \{i \in [1,n] \mid f'(\boldsymbol{x^i}) - f'(\boldsymbol{x}) > 0\}$ denote the 0 bit positions of $\boldsymbol{x}$ where the flipping can generate a solution with a positive increment on f'. Let $\delta = \min\{\frac{w_i}{f'(\boldsymbol{x^i}) - f'(\boldsymbol{x})} \mid i \in I\}$. Then, $\delta \leq OPT/(q-k)$. Otherwise, for any $e_i \in \boldsymbol{x}^* - \boldsymbol{x}$, $w_i > (f'(\boldsymbol{x^i}) - f'(\boldsymbol{x})) \cdot OPT/(q-k)$. Thus, $\sum_{e_i \in \boldsymbol{x}^* - \boldsymbol{x}} w_i > \big(\sum_{e_i \in \boldsymbol{x}^* - \boldsymbol{x}} (f'(\boldsymbol{x^i}) - f'(\boldsymbol{x}))\big) \cdot OPT/(q-k)$. Since f is monotone and submodular, f' is also monotone and submodular, then $f'(\boldsymbol{x}^*) - f'(\boldsymbol{x}) \leq \sum_{e_i \in \boldsymbol{x}^* - \boldsymbol{x}} (f'(\boldsymbol{x^i}) - f'(\boldsymbol{x}))$. Thus, $\sum_{e_i \in \boldsymbol{x}^* - \boldsymbol{x}} w_i > (f'(\boldsymbol{x}^*) - f'(\boldsymbol{x})) \cdot OPT/(q-k) = OPT$, which contradicts with $\sum_{e_i \in \boldsymbol{x}^* - \boldsymbol{x}} w_i \leq w(\boldsymbol{x}^*) = OPT$. Thus, by selecting $\boldsymbol{x}$ in step 5 of Algorithm 2 and flipping only the 0 bit corresponding to δ in step 6, it can generate a new solution $\boldsymbol{x'}$ with $\min\{q, f(\boldsymbol{x'})\} = k' > k$ and

$$w(\boldsymbol{x'}) \leq w(\boldsymbol{x}) + (k' - k) \cdot OPT/(q-k) \leq R_{k'} \cdot OPT.$$

Once generated, $\boldsymbol{x'}$ will be included into P. Otherwise, there must exist a solution in P dominating $\boldsymbol{x'}$ which has a larger f' and a smaller w; this implies that $K_{\max}$ has already been larger than k, which contradicts with the assumption $K_{\max} = k$. After including $\boldsymbol{x'}$, $K_{\max}$ increases from k to k'.

The probability of flipping a specific 0 bit of $\boldsymbol{x}$ is at least $\frac{1}{N+1} \cdot \frac{1}{n}(1 - \frac{1}{n})^{n-1} \geq \frac{1}{en(N+1)}$. Thus, the expected running time for such a step of increasing $K_{\max}$ is at most $en(N+1)$. Since N such steps are sufficient to make $K_{\max} = q$, the expected running time of this phase is $O(N^2 n)$.

By combining the two phases, the expected running time of the whole process is $O(Nn(\log n + \log w_{\max} + N))$. $\square$

For the penalty function method, we first show that minimum set cover (MSC) is an instance of minimum cost coverage and then give a concrete MSC example where the penalty function method is less efficient than Pareto optimization.

Definition 6 (Minimum Set Cover) *Given a set $S = \{e_1, \ldots, e_m\}$, a collection $C = \{C_1, \ldots, C_n\}$ of subsets of S with corresponding costs $w : C \to \mathbb{N}$, it is to find a subset of C (represented by $\boldsymbol{x} \in \{0,1\}^n$) with the minimum cost such that all the elements of S are covered, that is*

$$\arg\min_{x \in \{0,1\}^n} w(\boldsymbol{x}) = \sum\nolimits_{i=1}^{n} w_i x_i \quad s.t. \quad \bigcup\nolimits_{i:x_i=1} C_i = S.$$

Let $U = C$ and $f(\boldsymbol{x}) = |\bigcup_{i:x_i=1} C_i|$. Then f is monotone and submodular. Since $f(\boldsymbol{x}) \leq m$ and $|S| = m$, it is easy to verify that the MSC problem is an instance of minimum cost coverage (i.e., Definition 5) with $q = m$. Obviously, the parameters $c \leq 1$ and $N \leq m$ for MSC.

Friedrich et al. [2010] have analyzed the running time of the penalty function method on a specific MSC example, as shown in Lemma 4 (i.e., Theorem 8 in their paper). Thus, we have Theorem 4 by letting ϵ being a constant and $w_{\max} = 2^n$.

Lemma 4 *[Friedrich et al., 2010] Let $\delta > 0$ be a constant and $n^{\delta-1} \leq \epsilon < 1/2$. The expected running time of the penalty function method on a MSC example ($m = \epsilon(1-\epsilon)n^2$) for finding an approximation better than $((1-\epsilon)w_{\max})/\epsilon$ is exponential w.r.t. n.*

Theorem 4 *There exists a minimum cost coverage instance, where the expected running time of the penalty function method for finding a H_{cq}-approximate solution is exponential w.r.t. n, N and $\log w_{\max}$.*

5 Pareto Optimization vs. Greedy Algorithm

For the above two problems, there exist greedy algorithms which efficiently find an optimal solution [Edmonds, 1971] and a H_{cq}-approximate solution [Wolsey, 1982], respectively. Thus, a natural question is whether Pareto optimization can be better than greedy algorithms. We give a positive answer by analyzing a concrete instance of minimum set cover (also minimum cost coverage).

Example 1 *[Yu et al., 2012] As shown in Figure 1, the ground set S contains $m = k(k-1)$ elements, and the collection C contains $n = (k+1)(k-1)$ subsets $\{S_1^*, \ldots, S_{k-1}^*, S_{1,1}, \ldots, S_{1,k}, \ldots, S_{k-1,1}, \ldots, S_{k-1,k}\}$ with costs $\forall i : w(S_i^*) = 1 + 1/k^2$ and $\forall i, j : w(S_{i,j}) = 1/j$.*

For Example 1, the global optimal solution is $C_{global} = \{S_1^*, \ldots, S_{k-1}^*\}$ with cost $(k-1)(1+1/k^2)$, and $C_{local} = \{S_{1,1}, \ldots, S_{1,k}, \ldots, S_{k-1,1}, \ldots, S_{k-1,k}\}$ is a local optimal solution with cost $(k-1)H_k$. We show the performance of the greedy algorithm [Chvatal, 1979], the Pareto optimization method and the penalty function method on this example in Propositions 1-3, respectively. The greedy algorithm iteratively selects a subset from C with the smallest ratio of its cost and the number of newly covered elements, and its performance is directly from Proposition 1 in [Yu *et al.*, 2012].

Proposition 1 *[Yu et al., 2012] For Example 1, the greedy algorithm [Chvatal, 1979] finds C_{local}.*

Proposition 2 *For Example 1, the Pareto optimization method finds C_{global} in $O(n^2 \log n)$ expected running time.*

Proof. The objective vector $\boldsymbol{g}$ is implemented as $\big(w(\boldsymbol{x}), m - |\bigcup_{i:x_i=1} C_i|\big)$. By using the same proof as Lemma 2 except that $w_{\max} = 1 + 1/k^2$ and $P_{\max} \leq m+1$ in this example,

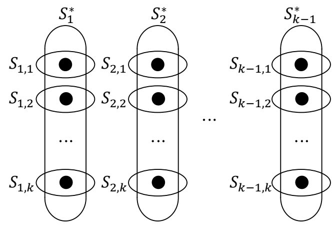

Figure 1: An example of the minimum set cover problem.

we can derive an upper bound $O(mn \log n) = O(n^2 \log n)$ on the expected running time for finding $\{0\}^n$.

Then, we are to show that starting from $\{0\}^n$, it can find the subset of $C_{global} = \{S_1^*, \ldots, S_{k-1}^*\}$ with size from 0 to $k-1$ sequentially. Note that any subset of C_{global} is Pareto optimal, since the cost of covering k elements using subsets only from $\{S_{i,j}\}$ is larger than $1 + \frac{1}{k^2}$. After finding $\boldsymbol{x} \subseteq C_{global}$ with $|\boldsymbol{x}| = i$, it can generate $\boldsymbol{x'} \subseteq C_{global}$ with $|\boldsymbol{x'}| = i+1$ in one iteration with probability at least $\frac{k-1-i}{(m+1)n}(1-\frac{1}{n})^{n-1} \geq \frac{k-1-i}{e(m+1)n}$, since it is sufficient to select $\boldsymbol{x}$ and flip one 0 bit corresponding to any unselected S_i^*. Thus, the expected running time to find the optimal solution C_{global} is at most $\sum_{i=0}^{k-2} \frac{e(m+1)n}{k-1-i} \in O(n^2 \log n)$. □

Proposition 3 *For Example 1, the penalty function method finds C_{global} in $O(n^3 \log n)$ expected running time.*

Proof. Let u denote the number of uncovered elements of the current solution. Obviously, u will not increase. Since each uncovered element has a corresponding $S_{i,j}$, flipping the corresponding bit of $S_{i,j}$ can decrease u by 1. Thus, u can decrease by 1 in one iteration with probability at least $\frac{u}{n}(1-\frac{1}{n})^{n-1} \geq \frac{u}{en}$. Hence, for finding a set cover (i.e., u decreases to 0), the expected running time is at most $\sum_{u=m}^{1} \frac{en}{u} \in O(n \log n)$. Note that after finding a set cover, the solution will always keep being a set cover.

Then, we apply Lemma 1 to derive the running time for finding C_{global}. Let $X_t = w(\boldsymbol{x^t}) - w_{opt}$, where $\boldsymbol{x^t}$ is the solution after t iterations of Algorithm 2 and w_{opt} is the optimal weight, i.e., $w(C_{global})$. Then, $X_t = 0$ implies that C_{global} is found, and T is just the running time for finding C_{global}.

We are to analyze $\mathbb{E}[\![X_t - X_{t+1} | X_t]\!]$. It is easy to see that $w(\boldsymbol{x^{t+1}}) \leq w(\boldsymbol{x^t})$. For $\boldsymbol{x^t}$, there are three covering cases for each column of Figure 1. For case (1) which contains S_i^* and several $S_{i,j}$, by deleting one $S_{i,j}$, $w(\boldsymbol{x^{t+1}}) = w(\boldsymbol{x^t}) - 1/j$ with probability $\frac{1}{n}(1-\frac{1}{n})^{n-1}$. For case (2) which contains all $S_{i,j}$ but not S_i^*, by including S_i^* and deleting $S_{i,1}$ and another $S_{i,j}$, $w(\boldsymbol{x^{t+1}}) = w(\boldsymbol{x^t}) - (1/j - 1/k^2)$ with probability $\frac{1}{n^3}(1-\frac{1}{n})^{n-3}$; by including S_i^* and deleting $S_{i,1}$ and any two from $\{S_{i,2}, \ldots, S_{i,k}\}$, $w(\boldsymbol{x^{t+1}}) < w(\boldsymbol{x^t}) - (k-2)/k^2$ with probability $\frac{\binom{k-1}{2}}{n^4}(1-\frac{1}{n})^{n-4}$. For case (3) which contains only S_i^*, it reaches the optimal case. By unifying these probabilities to a lower bound $\frac{k-2}{2e(k+1)n^3}$, the sum of the improvements is just $w(\boldsymbol{x^t}) - w_{opt}$. Thus,

$$\mathbb{E}[\![w(\boldsymbol{x^{t+1}})]\!] \leq w(\boldsymbol{x^t}) - \frac{k-2}{2e(k+1)n^3}(w(\boldsymbol{x^t}) - w_{opt}).$$

Then, we have $\mathbb{E}[\![X_t - X_{t+1} | X_t]\!]$

$$= w(\boldsymbol{x^t}) - \mathbb{E}[\![w(\boldsymbol{x^{t+1}})]\!] \geq \frac{k-2}{2e(k+1)n^3} \cdot X_t.$$

By Lemma 1, $s_0 < k^2$ and $s_{\min} \geq 1/k$, we have

$$\mathbb{E}[\![T | X_0]\!] \leq \frac{2e(k+1)n^3}{k-2} \cdot (1 + 3\log k) \in O(n^3 \log n).$$

□

6 Conclusion

Pareto optimization employs a bi-objective optimization as an intermediate step for solving constrained optimization problems. Recent application [Qian *et al.*, 2015] has shown its theoretical and empirical advantages. This work further theoretically investigates its optimization ability. Pareto optimization is compared with the widely used penalty function method on two large problem classes, the P-solvable minimum matroid optimization and the NP-hard minimum cost coverage problem. On both problems, Pareto optimization is proved to be more efficient. Moreover, Pareto optimization is shown to be better than the greedy algorithm. With these theoretical results, we expect that Pareto optimization will attract more attentions for constrained optimization tasks.

References

[Auger and Doerr, 2011] A. Auger and B. Doerr. *Theory of Randomized Search Heuristics - Foundations and Recent Developments*. World Scientific, Singapore, 2011.

[Bäck, 1996] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996.

[Ben Hadj-Alouane and Bean, 1997] A. Ben Hadj-Alouane and J. C. Bean. A genetic algorithm for the multiple-choice integer program. *Operations Research*, 45(1):92–101, 1997.

[Bertsekas, 1999] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Cambridge, MA, 1999.

[Cai and Wang, 2006] Z. Cai and Y. Wang. A multiobjective optimization-based evolutionary algorithm for constrained optimization. *IEEE Transactions on Evolutionary Computation*, 10(6):658–675, 2006.

[Chvatal, 1979] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[Coello Coello, 2002] C. A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11):1245–1287, 2002.

[Cormen *et al.*, 2001] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.

[Deb *et al.*, 2002] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[Deb, 2000] K. Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186(2):311–338, 2000.

[Doerr *et al.*, 2012] B. Doerr, D. Johannsen, and C. Winzen. Multiplicative drift analysis. *Algorithmica*, 64(4):673–697, 2012.

[Edmonds, 1971] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1(1):127–136, 1971.

[Friedrich *et al.*, 2010] T. Friedrich, J. He, N. Hebbinghaus, F. Neumann, and C. Witt. Approximating covering problems by randomized search heuristics using multi-objective models. *Evolutionary Computation*, 18(4):617–633, 2010.

[He and Yao, 2001] J. He and X. Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127(1):57–85, 2001.

[Homaifar *et al.*, 1994] A. Homaifar, C. X. Qi, and S. H. Lai. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–253, 1994.

[Kempe *et al.*, 2003] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 137–146, Washington, DC, 2003.

[Korte and Vygen, 2012] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, Berlin, Germany, 2012.

[Krause *et al.*, 2008] A. Krause, B. McMahan, C. Guestrin, and A. Gupta. Robust submodular observation selection. *Journal of Machine Learning Research*, 9(12):2761–2801, 2008.

[Laumanns *et al.*, 2004] M. Laumanns, L. Thiele, and E. Zitzler. Running time analysis of multiobjective evolutionary algorithms on pseudo-Boolean functions. *IEEE Transactions on Evolutionary Computation*, 8(2):170–182, 2004.

[Michalewicz and Schoenauer, 1996] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[Nemhauser *et al.*, 1978] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Mathematical Programming*, 14(1):265–294, 1978.

[Neumann and Wegener, 2007] F. Neumann and I. Wegener. Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theoretical Computer Science*, 378(1):32–40, 2007.

[Neumann and Witt, 2010] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization - Algorithms and Their Computational Complexity*. Springer-Verlag, Berlin, Germany, 2010.

[Qian *et al.*, 2013] C. Qian, Y. Yu, and Z.-H. Zhou. An analysis on recombination in multi-objective evolutionary optimization. *Artificial Intelligence*, 204:99–119, 2013.

[Qian *et al.*, 2015] C. Qian, Y. Yu, and Z.-H. Zhou. Pareto ensemble pruning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 2935–2941, Austin, TX, 2015.

[Venkatraman and Yen, 2005] S. Venkatraman and G. G. Yen. A generic framework for constrained optimization using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9(4):424–435, 2005.

[Wolsey, 1982] L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

[Yu and Zhou, 2008] Y. Yu and Z.-H. Zhou. A new approach to estimating the expected first hitting time of evolutionary algorithms. *Artificial Intelligence*, 172(15):1809–1832, 2008.

[Yu *et al.*, 2012] Y. Yu, X. Yao, and Z.-H. Zhou. On the approximation ability of evolutionary optimization with application to minimum set cover. *Artificial Intelligence*, 180:20–33, 2012.

[Zhou and He, 2007] Y. Zhou and J. He. A runtime analysis of evolutionary algorithms for constrained optimization problems. *IEEE Transactions on Evolutionary Computation*, 11(5):608–619, 2007.