Generating Combinations on the GPU and its Application to the K-Subset Sum

Victor Parque
Department of Modern Mechanical Engineering
Waseda University, Japan
parque@aoni.waseda.jp

ABSTRACT

Efficiently representing and generating combinations can allow the seamless visualization, sampling, and evaluation of combinatorial architectures. In this paper, being relevant to tackle resource allocation problems ubiquitously, we address the subset sum problem by (1) using gradient-free optimization with a number-based representation of the combinatorial search space and by (2) generating combinations with minimal change order through parallel reductions in the GPU.

Our computational experiments consisting of a relevant set of problem instances and gradient-free optimization algorithms show that (1) it is possible to generate combinations in the GPU efficiently, with quasi-linear complexity, (2) it is possible to tackle instances of the subset sum problem within a reasonable number of function evaluations, and (3) Particle Swarm Optimization with Fitness Euclidean Ratio converges faster.

Since the search space of number-based representations is one-dimensional and amenable to parallelization schemes (e.g., GPU), we believe our work opens the door to tackle further combinatorial problems.

CCS CONCEPTS

• **Computing methodologies** → **Search methodologies**; • **Applied computing** → **Operations research**;

KEYWORDS

Subset Sum, Knapsack Problem, Number Representation, Enumerative Encoding, Combinations, Gradient-Free, Differential Evolution, Particle Swarm, Optimization, GPUs, Parallel Reduction

ACM Reference Format:
Victor Parque. 2021. Generating Combinations on the GPU and its Application to the K-Subset Sum. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449726.3463226

1 INTRODUCTION

The research of combinatorial problems allows designing complex behaviors and systems ubiquitously, particularly in Reliability Engineering[1], in Decision Sciences[2], in Operations Research[3], in Graph Theory[4], in Cryptography[5], in Database Management Systems[6], in Network Design[7–9], in Material Science[10], in Chemistry[11], in Logic Systems [12] and in Language Modeling[13].

The subset-sum problem aims to find a subset from a superset meeting a user-defined aggregation value, such as the sum of its elements. This problem is relevant to resource allocation problems, and its study is particularly important to discrete design, operations research and management fields. This paper tackles a class of the subset-sum problem where the subset is of fixed size. This problem instance is a particular case of the more general, widely-studied Knapsack Problem (KP) and the Subset-Sum Problem (SSP)[14] which are known to be NP-hard problems[15].

The subset-sum problem is an NP-hard problem as no polynomial-time algorithm exists to solve it [16, 17]. Dynamic Programming[18–20] and exact algorithms extending the Bellman's recursion exists for small-scale instances of the subset-sum problem[21]. Moreover, the 3/4 approximation ratio is a well-known worst-case guarantee [14, 22]. As such, heuristics search methods[23, 24], as well as local search methods[25] are the usual preference. The reason behind this choice is the compelling approximation benefits and the fast response time. Also, parallel computing often improves time and space bounds wherever feasible; for instance, the parallel version of the branch and bound algorithm[27] and the efficient implementations of the parallel-random machine model[28].

Branch and bound[29] and genetic algorithms [30, 31] are well-established global search methods tackling the subset-sum problem. The appealing nature of being gradient-free and flexible allows for tailored selection, particular data structures, and local search schemes. For instance, evolutionary computing with the penalty selection method[32], genetic algorithms using the rejection of infeasible offsprings [33], and a tailored penalty method [34]. For instance, the dynamic evolutionary optimization of the subset-sum problem [35], the dynamic multi-objective optimization with a genetic algorithm and the external archive, and a hybrid between a Pareto dominance and aggregated fitness [36], and the (1+1) evolutionary algorithms with the superiority of feasible point selection and the binary representation[37].

Methods for sampling, mutation, and selection tailored to the subset-sum problem received relevant attention in the community. Often, the binary and the tuple representations are the usual choices. For a superset of size n and a subset of size k, the binary representation is an n-dimensional binary vector space with k ones and $n-k$ zeros; and the tuple representation is a k-dimensional vector related to a combination element out of the $\binom{n}{k}$. The subset-sum with fixed size under rather distinct representation than the binary and the tuple representation has been elusive in the literature. The essential

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463226

role of effectively representing the search space in the subset-sum problem has been discussed in [35, 38]. Moreover, [39] discussed the possibility of using the number-based representation and Differential Evolution with the global and local neighborhood, and [40] compared the feasibility of using nature-inspired algorithms. This paper extends [39] and [40] by enabling parallel reductions in the GPU to generate associated combinations from numbers; thus, gradient-free optimization algorithms can sample the number search space more efficiently, allowing further scalability on n. In particular, our contributions are as follows:

- We tackle the subset sum problem with fixed size over the integer search space, rendering a one-dimensional combinatorial search space. Compared to the conventional representations, our approach is amenable to parallelization schemes.
- We evaluate the generation of combinations from numbers using a decoding algorithm based on revolving door order through a Graphics Processing Unit. Our theoretical observations and experiments show the efficiency quasi-linear complexity, which is good due to its implications for scalability.
- We evaluate a relevant set of nature-inspired optimization algorithms considering distinct forms of gradient-free sampling, selection pressure, the balance of exploration-exploitation, and the multimodality considerations. Our computational experiments show the attainability of obtaining optimal solutions within a reasonable number of evaluations (10^4).

2 PROPOSED APPROACH

2.1 The Subset-Sum Problem

Let a superset be $\mathcal{S} = [n] = \{1, 2, 3, ..., n\}$, a k-subset of $[n]$ is the tuple $c = (c_1, c_2, ..., c_k)$, with $c_i \in [n]$.

The subset-sum problem with a fixed size aims to

$$\textbf{Find } c \text{ such that } \sum_{i=1}^{k} c_i = s, \tag{1}$$

where c_i is the ith element of the tuple c, k is the size of the combination object and s is a user-defined aggregation value.

Generalizing (1) for variable size k leads to the widely-studied Knapsack Problem. And when (1) is a weighted sum, the above formulation becomes the balanced subset-sum[41].

The reader may note that $\mathcal{S}$ refers to the set of integer numbers up to n. By studying the possible values of combination elements, it is possible to compute the feasible limits in the user-defined sum s. The combination with the smallest integers is $c = (1, 2, 3, ..., k)$, thus the lower bound of the sum s becomes the sum of such numbers

$$s_l = \frac{k(k+1)}{2}. \tag{2}$$

Conversely, the combination with the largest integer numbers from the set $\mathcal{S}$ is $c = (n-k+1, n-k+2, ..., n-2, n-1, n)$, then the upper bound of the sum s is the sum

$$s_u = n.k - \frac{k(k-1)}{2}. \tag{3}$$

Let

$$C = \left\{C_1, C_2, ..., C_{\binom{n}{k}}\right\} \tag{4}$$

be the $\binom{n}{k}$ different k combinations of the set $[n]$, with $n \geq k$, and $k \geq 1$.

It is possible to represent general combinations by using the k-tuple $(c_1, c_2, ..., c_k)$[45] and by using the *binary* n-tuple $(b_1, b_2, ..., b_n)$ in which k ones and $n-k$ zeros correspond to selected and unselected elements of the set $\mathcal{S}$.

In this paper, we use an alternative method to represent a combination: by using the integer numbers. Here, each element of the set C is associated with an integer number from the set $\mathbb{I} = \{0, 1, 2, ..., \binom{n}{k} - 1\}$, thus there exists a bijection $f : \mathbb{I} \rightarrow C$, that generates a combination tuple c given an integer number[42–44].

Accordingly, we tackle (1) by formulating the subset-sum problem with fixed size as an optimization problem over the integer search space

$$\underset{x}{\textbf{Minimize}}\ F(x) \text{ subject to } x \in \mathbb{I}, \tag{5}$$

in which the cost function F is as follows (example in Fig. 1-(a)):

$$F(x) = \left| g(f(x)) - s \right| \tag{6}$$

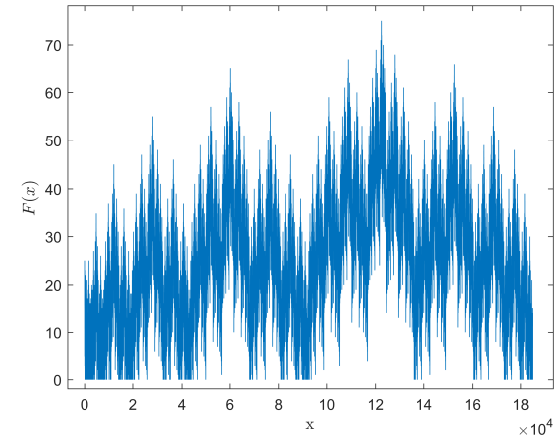
$$g(c) = \sum_{i=1}^{k} c_i, \tag{7}$$

where $|...|$ in (6) denotes the absolute function, x is the integer representing the combination tuple c, $f(x)$ is the function which generates the combination tuple given the integer number $x \in I$, s is the user-defined subset-sum s given the boundary $\in [s_l, s_u]$), and $g(.)$ is the aggregation function of the elements of the combination c.

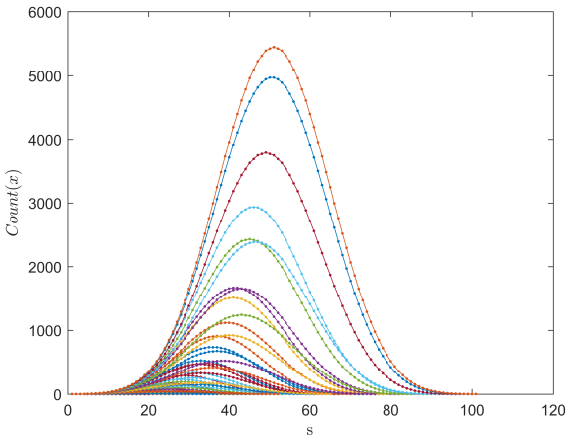
Since searching over the integer search space $x \in I$ implies generating big numbers when n and k becomes relatively large, we use the absolute value to avoid issues in bloating. The proposed cost function F in (6) is non-differentiable and multi-modal:

- Both the absolute value in (6) and the aggregation function g imply non-differentiable terms, thus gradient-free optimization algorithms over the search space I are suitable to tackle problem (5). For instance, Fig. 1-(a) shows and example of the landscape of the cost function $F(x)$ for all combinations (by complete enumeration) associated to the particular case $x \in I$ and $n = 20$, $k = 10$, $s = 80$. The reader may easily the non-convexity and the rugged fitness landscape.
- Also, both the absolute value in (6) and the aggregation function g imply multi-modality, that is the possibility to find multiple optimal solutions $x^* \in I$ that meet the criterion $F(x^*) = 0$. For example, Fig. 1-(b) shows the histogram (frequency) of the count of the number of optimal solutions fulfilling $F(x^*) = 0$ for all combinations within the particular case $n = [1, 20]$, $k = [1, n/2]$, $s \in [s_l, s_u]$. By looking at Fig. 1-(b), the bell-shaped distribution shows that it is possible to find more than one optimal solution.

In line with the above, some of the relevant motivations to study the number-based representation in tackling the subset-sum problem lies in the following notions



(a) $F(x)$



(b) Histogram($x*$)

Figure 1: (a) The cost function $F(x)$ for all the combinations with $n = 20$, $k = 10$, $s = 80$. (b) Count of optimal solutions x^* satisfying $F(x) = 0$ for all $n = [1, 20]$, $k = [1, n/2]$, $s \in [s_l, s_u]$.

- Since each integer number in the set $\mathbb{I}$ represents a unique combination element of C, the number-based search space defined by $x \in I$ is complete, is information-theoretically optimal, and is canonical.
- Rather than being k-dimensional or n-dimensional, searching over the space of numbers $x \in I$ becomes a one-dimensional problem, whose landscape is attractive for gradient-free optimization algorithms.
- It is also possible to sample the search space of the subset-sum with equal probability: generating a combination from a number is independent of the past generation history of combinations.
- For large n, using the integer domain is preferable when it is impossible to generate the full set of combinations[45–52], and when it is impractical to generate combinations randomly [53, 54] and sequentially [55].

Algorithm 1 Unranking Algorithm

1: **procedure** UNRANK(x, k)
2: **Input** x ▷ Rank Number
3: **Output** $(c_1, c_2, ..., c_k)$ ▷ Combination Object
4: $c_k^o \leftarrow k$ ▷ Initial approximate solution
5: **for** $i \leftarrow k$ **downto** 1 **do**
6: $c_i \leftarrow 1 + \left\lfloor \text{Minimize } |J(c, i, x)| \text{ with } c \geqslant i \right\rfloor$
7: $x \leftarrow \binom{c_i}{i} - x - 1$
8: **end for**
9: **return** $(c_1, c_2, ..., c_m)$ ▷ Combination object
10: **end procedure**

2.2 Generating Combinations on the GPU

To realize the function $f : \mathbb{I} \rightarrow C$ being able to decode an integer number to render a combination tuple c, we use a decoding algorithm based on the revolving door order as shown by Algorithm 1, in which the element c_i is generated by minimizing the cost function $|J(c, i, x)|$ for variable c and constraint $c \geq i$.

In algorithm 1,

$$J(c, i, x) = \left\lceil \sum_{p=1}^{i} \log_b \left(\frac{c - i}{p} + 1 \right) \right\rceil - \log_b(x), \tag{8}$$

where i denotes the index in the *for* loop of Algorithm 1 ($i = k$ at initial iteration), x denotes the integer number representing the combination, and the base b in $\log_b(.)$ is a user-defined constant (the larger the coefficient $\binom{n}{k}$ is, the larger the constant b to handle big numbers accurately). For standard computing environments, we use $b = 10$ since it allows to compute $\log_b(g)$ in feasible range.

To find the minimal of $|J|$ in $c \in [i, +\infty]$, we used a gradient-based scheme, as follows:

$$c_i^{k+1} = \begin{cases} c_i^k - \frac{J}{J'}, & \text{if } c_i^{k+1} \geqslant i \\ i, & \text{otherwise} \end{cases} \tag{9}$$

where $i \in [m]$, k denotes the iteration number, and the subscript J' denotes the first derivative of the function J with respect to c_i^k.

$$J'(c, i) = \frac{1}{\ln(b)} \sum_{p=0}^{i-1} \frac{1}{(c - p)} \tag{10}$$

Due to the concavity of the function J in $c \in [i, +\infty]$, the initial solutions $c_i^o \in [i, +\infty]$ ensure convergence to the root of J, and are initialized as follows:

$$c_i^o = \begin{cases} m, & \text{if } i = m \\ c_{i+1}, & \text{otherwise} \end{cases} \tag{11}$$

The above is based on the revolving door ordering principle: $c_1 < c_2 < ... < c_m$, and the closeness of c_i to c_{i+1}, which ensures efficient convergence to the global optima. Thus, the *minimal change*

ordering is useful when sampling properties of consecutive combinations efficiently. It is possible to use different orderings, e.g., lexicographical ordering. However, the *minimal change ordering* is preferable to preserve distance when sampling arbitrary solutions in the integer search space: the phenotypic difference of combinations associated with consecutive integer numbers is one.

As for termination criterion, we use the following criteria

$$|J'| < \delta, \tag{12}$$

$$|c^{k+1} - c^k|/|c^k| < \varepsilon \tag{13}$$

in which δ and ε are threshold tolerances to avoid division and sampling with very small numbers. Without loss of generality within the context of standard desktop environments, we use $\delta = \varepsilon = 10^{-8}$.

Since both functions J and J' in Eq. 11 are parallelizable and computable in $O(i)$ time by using a single processor, and in $O(\log i)$ time by using at most $O(i/\log i)$ processors (to ensure work-efficiency in parallel cores according to the Brent's Theorem[1]), we used a parallel reduction algorithm. As such, we reduce i sums in multiple elements per thread in a CUDA-enabled Graphics Processing Unit (GPU).

2.3 Complexity of Decoding Combinations

To evaluate the computational efficiency of generating combinations, we define the asymptotic complexity behavior as the function:

$$T(k) \approx \underbrace{\sum_{l=1}^{k}\sum_{j=1}^{t_l}\alpha(l)}_{F(k)} + \underbrace{\sum_{l=1}^{k}\beta(l)}_{G(k)}, \tag{14}$$

where l is the order of the **for** loop in Algorithm 1[2]; t_l represents the number of iterations used to solve the minimization problem (in line 6 of algorithm 1) in the l-th **for** loop; $\alpha(l)$ denotes the complexity of evaluating the objective function J and its gradient J' during the l-th **for** loop; and $\beta(l)$ denotes the time complexity of evaluating the binomial coefficient of line 7. Basically, $T(k)$ computes the expected complexity to solve the k minimization problems and k binomial coefficients, denoted by $F(k)$ and $G(k)$, respectively.

- In case of using a single processor, the time complexity of both $\alpha(l)$ and $\beta(l)$ is $O(i)$. Then,

$$F(k) \approx kt_1 + (k-1)t_2 + (k-2)t_3 + ... + t_k \tag{15}$$

$$G(k) \approx k^2 \tag{16}$$

- In case of using at most $O(k/\log k)$ processors, the time complexity of both $\alpha(l)$ and $\beta(l)$ is $O(\log i)$. Then,

$$F(k) \approx \log(k)t_1 + \log(k-1)t_2 + ... + \log(1)t_k \tag{17}$$

$$G(k) \approx k\log(k) \tag{18}$$

[1] assuming algebraic operations with numbers in $O(1)$
[2] $l = 1$ when $i = k$, and $l = k$ when $i = 1$

3 COMPUTATIONAL EXPERIMENTS

This section describes our experiments and results evaluating the feasibility of our proposed approach.

3.1 Generating Combinations

To evaluate the feasibility to generate combinations in the GPU, we implemented the algorithm 1 in CUDA, and experimentally investigated the complexity behaviour and the number of iterations t_l used for decoding combinations from given integer numbers. Our computing environment consisted of an Intel i7-4930K @ 3.4GHz, and algorithms were implemented in Matlab.

We generated combinations from the first billion group from values of $n \in \{100, 200, ..., 1000\}$ and $k \in \{100, 200, ..., 1000\}$ in our computing environment. The main reason behind using values of n up to 1000 is due to the fact of having computational restrictions on the allowable representation of integer numbers (the maximum number being able to be represented in a standard computing environment in Matlab is defined by realmax, which is 1.7977×10^{308}). However, we argue that it is possible to overcome this limit by numerical precision libraries and extend the memory capacity of our computing environment, which is out of the scope of this paper. Naturally, since our approach is useful for applications involving arbitrary (non-sequential) sampling without repetition, compression efficiency, and concurrency (parallelization), our future work would aim at evaluating the accuracy and feasibility of using advanced computing environments (e.g., multi CPU-GPU devices).

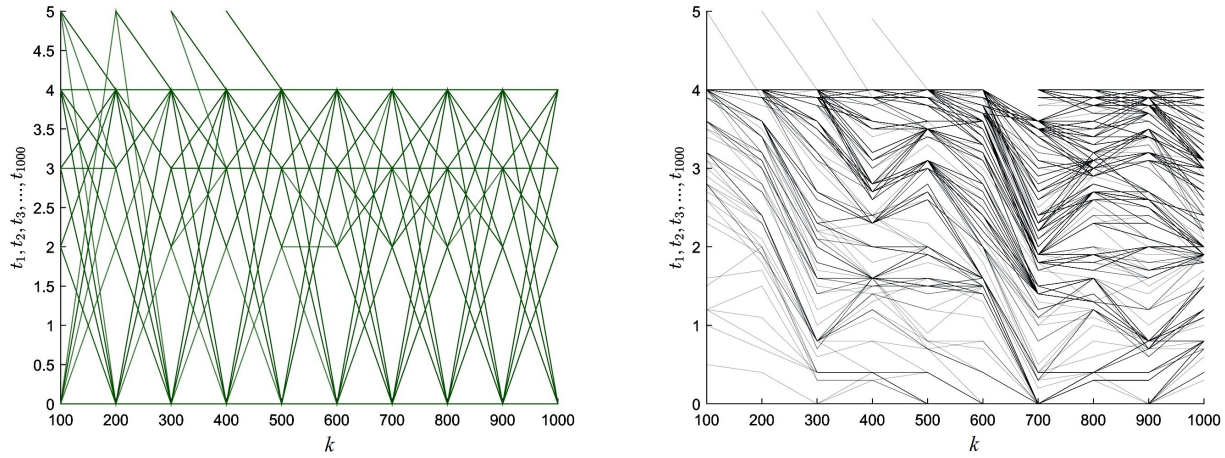
Fig. 2 portrays the total number of iterations for all n and k, and Fig. 3 estimates complexity metrics with respect to the value of k. The reason we evaluate both Fig. 2 and Fig. 3 as a function of k is due to algorithm 1 is a function of k for a given x. As we can observe from Fig. 2, the number of iterations t_l is constant, in the worst case, and the average number of iterations decreases as a function of k. Thus, assuming that $t_1 = t_2 = ... = t_l = ... = t_k = t$ in (15)-(17), the time complexity is expected to be bounded by $O(m^2)$, when using a single processor (from Eq. 15), and bounded by $O(k \log k)$, when using at most $O(k/\log k)$ processors (from Eq. 17).

The above-mentioned observations agree with our empirical estimations of Fig. 3 which shows the complexity metrics $T(k)$ when using a single and multiple processors. In Fig. 3 - Fig. 4, we can observe that a quasi-quadratic function is obtained when using one processor, and a quasi-linear behaviour is achieved when using more than one processor. Furthermore, the above attained complexity is independent of the number of elements n, which is scalable when n is large, or time-varying.

3.2 Gradient-free Optimization Algorithms

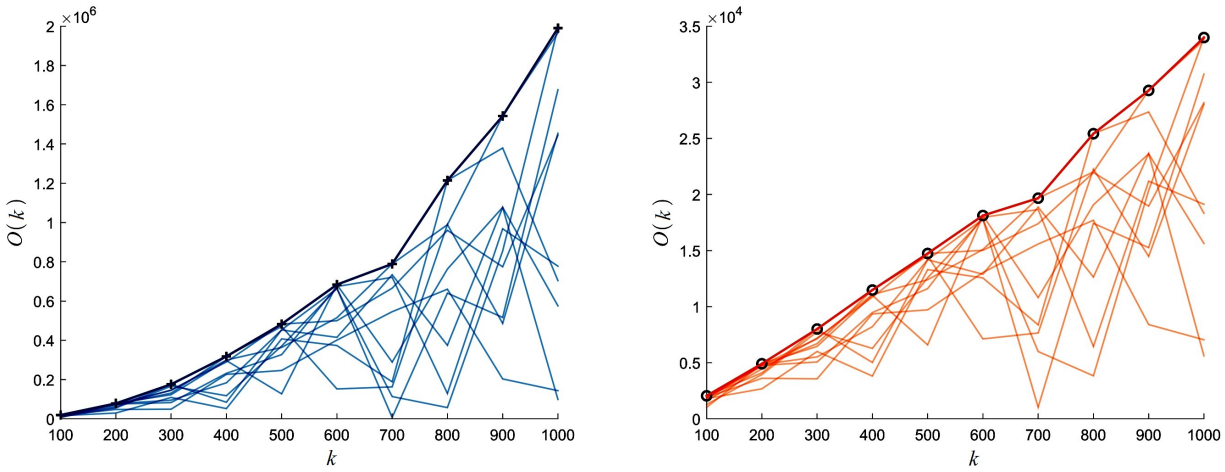
We used the following heuristics:

- Particle Swarm Optimization with Fitness Euclidean Ratio (FERPSO)[56],
- Differential Evolution with Successful Parents (DESPS)[57],
- Dividing Rectangles Algorithm (DIRECT)[58, 59],
- Rank-Based Differential Evolution (RBDE)[60],
- Success-History Based Parameter Adaptation Differential Evolution (SHADE)[61],
- Differential Evolution with Global and Local Neighborhoods (DEGL)[62],



(a) Number of iterations (b) Average number of iterations

Figure 2: Number of iterations when generating combinations in $n, k \in [100, 1000]$.



(a) One processor (b) Many processors

Figure 3: Complexity as a function of k when generating combinations in $n, k \in [100, 1000]$.

Our motivation for using the above set is to include distinct forms of sampling, selection pressure, the balance of exploration-exploitation, and multi-modality considerations. Although some of the algorithms mentioned above perform in continuous search spaces, the rounding function is a straightforward mechanism that enables through comparisons across experiments. Devising a sampling scheme tailored to integer numbers is potential to improve convergence further and is left for future work in our agenda.

Some of the key parameters in the above-mentioned schemes are inertia weight $\omega = 0.7$, weight on the local best $c_1 = 0.5$, weight on the global best $c_2 = 1$, population size 100. In Differential Evolution algorithms, the crossover rate was $CR = 0.5$, the mutation strategy was DE-current-to-best/1 mutation strategy with scaling factors $F = 0.7$. The neighborhood ratio in DEGL $\lambda = |\frac{ln(U(0,1))}{2}|$. The coefficient β involved in the Whitley distribution scheme in Rank-Based Differential Evolution (RBDE) was set as $\beta = 2$. The global/local weight parameter in DIRECT was set to 10^{-4}. Fine tuning of these parameters is out of the scope of this paper.

3.3 Tackling the Subset Sum

We used the following settings for subset sum problems: $n = 50$, and $k = n/2$, in which

- Values of user-defined subset sum s are defined as:

$$s = (1 - a).s_l + a.s_u, \tag{19}$$

for $a = \{0.25, 0.75\}$.

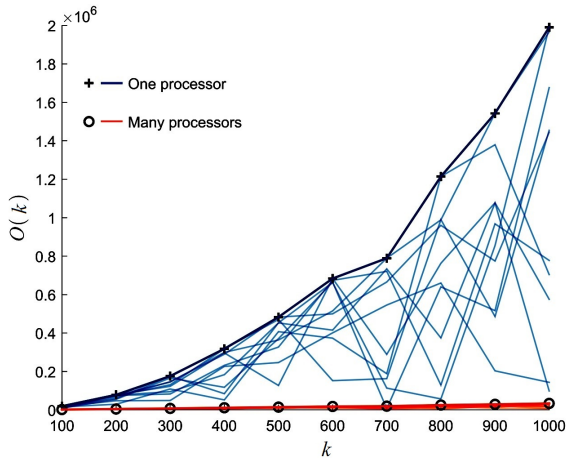


Figure 4: Comparison of complexity metric as a function of k when generating combinations in $n, k \in [100, 1000]$.

- For each value a, 30 runs were performed to solve Eq. 5.
- For each run the maximum number of evaluations was 10^4.

The principal motivation of using 10^4 evaluations is to evaluate the efficiency of the proposed approach under a restrictive evaluation budget. Moreover, using multiple independent runs allows fair comparison over random initialization schemes. Also, by setting $a = \{0.25, 0.75\}$, we enable to evaluate fitness landscapes distributed in $[s_l, s_u]$. Using $a = 0$ and $a = 1$ are trivial, since they correspond to the combinations $(1, 2, ..., k)$ and $(n-k+1, n-k+2, ..., n-1, n)$, respectively. Furthermore, for an extended search space, we set the value of $k = n/2$ for a wide search space. Since the solutions are encoded by $x \in \mathbb{I}$, for $\mathbb{I} = \{0, 1, 2, ..., \binom{n}{k} - 1\}$ and $|\mathbb{I}| = \binom{n}{k}$, then $k = n/2$ allows $|\mathbb{I}| = \binom{n}{n/2}$ to attain the largest size for any given value of n. For example, when $n = 50$ and $k = 25$, the search space consists of $|\mathbb{I}| = 1.26 \times 10^{14}$ numbers, which is a challenging search space.

3.4 Results and Discussion

Fig. 5 shows the mean convergence of the cost function $F(x)$ across the best solutions over independent runs, whereas Fig. 6 shows its standard deviation. In these figures, the x-axis depicts the number evaluations, and the y-axis shows the cost function F. The global optima relates to the criterion $F = 0$. The reader may note that in each plot of Fig. 5, there exists six convergence lines, each of which related to a gradient-free optimization algorithm. Although, some of the studied algorithms, such as FERPSO, are suitable for multi-modal problems, we obtain the single/best optimal solution for a fair comparison. By observing Fig. 5 and Fig. 6, we observe note the following facts:

- Among the studied algorithms, DIRECT shows sudden decreases over a number of function evaluations. This is due to DIRECT being a deterministic algorithm, thus the standard deviation in Fig. 6 is zero across independent runs. Although it is possible to a stochastic-like feature by allowing variable grid formation, the study of such extensions is left for future agenda.
- Among the studied algorithms, FERPSO shows the faster average convergence (over 30 runs). We believe this observation occurs due to the sparsity-inducing mechanisms to sample close and elite solutions within multiple (implicit) neighborhoods. Since the fitness landscape is noisy and multimodal, Fig. 1-(a) shows, FERPSO naturally suits to tackle the global optima with faster convergence. As such, we observed that FERPSO achieved the global optima in around 3000 function evaluations, whereas other algorithms required between 6000 and 9000 function evaluations, in the best cases.
- The search space is wide: it has more than 10^{14} numbers. Despite such a large space, most of the studied algorithms converged to the global optima or close to it. Also, as the standard deviation decreased with respect to the number of evaluations, the population of nature-inspired algorithms converged to the global optima, or close to it.
- The above observations imply the feasibility of using gradient-free optimization algorithms over the integer space. The fact of using an enumerative representation with a minimal change order enables to generate a sparse fitness landscape over the entire combinatorial search space. In some particular situations, we also observed that random initialization was sufficient to generate optimal or close to optimal solutions to the subset sum problem. Investigating this phenomenon for large n is in our future agenda.

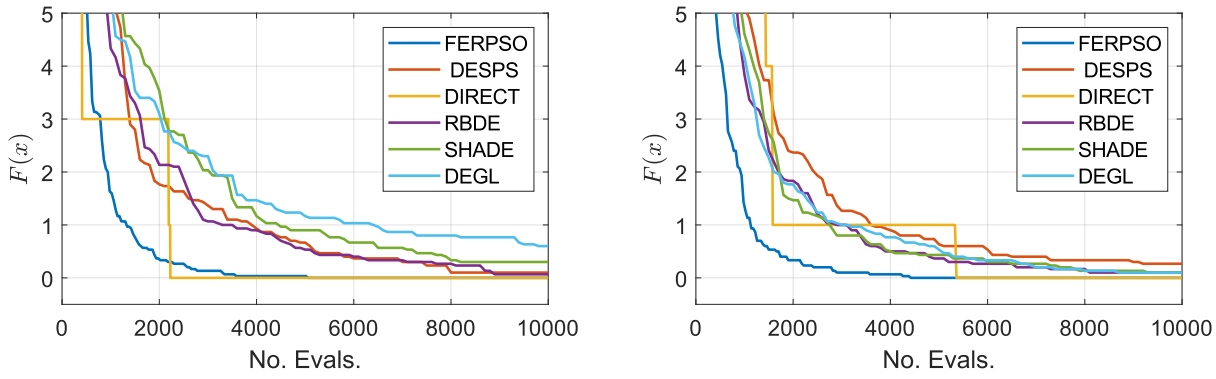
Furthermore, we compared the statistical difference in the converged cost functions after 10^4 evaluations. Fig. 7 shows the statistical significance test over 30 independent runs based on the Wilcoxon test at 5% significance level. In this figure, the symbols with '+/=/-' denote situations in which an algorithm in the row (vertical axis of the heat map) is significantly better/similar/worse to an algorithm in the column (horizontal axis of the heat map). By looking at the obtained results, we can confirm that, most algorithms, with the exception of DEGL, perform similarly by achieving comparatively similar performance.

The observations mentioned above imply that the highly sparsity-inducing mechanisms induced by neighborhood strategies such as the Particle Swarm with Fitness Euclidean Ratio (FERPSO)[56] are potential for fast convergence, in average. Moreover, most gradient-free optimization algorithms studied in this paper converged to optimal solutions, despite tackling a large search space consisting of about 10^4 function evaluations. We believe the above-described phenomenon is due to the integer search space and the minimal change order tracking the potential solutions in the combinatorial search space. The further study of tailored mutation strategies and optimization algorithms is on our agenda.

We believe our results in this paper offer the building blocks to extend the nature-inspired optimization algorithms using enumerative encoding. Investigating the performance over a large number of combinatorial optimization problems, based on the implications of the subset sum problem and the GPU-based generation of combinatorial objects, is on our agenda.

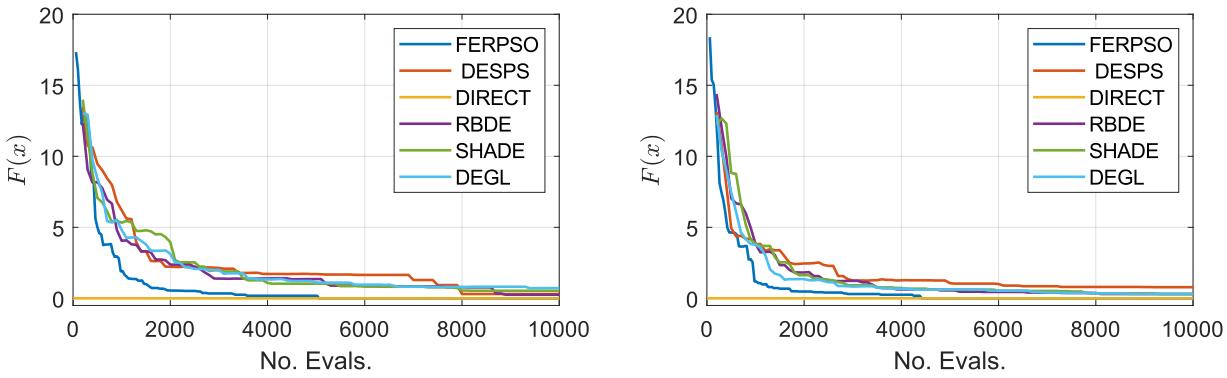
4 CONCLUSIONS

We have presented a new approach for the subset sum problem using gradient-free optimization. Our approach samples the number



(a) $a = 0.25$ (b) $a = 0.75$

Figure 5: Mean convergence behaviour of the evaluated algorithms over 30 independent runs for $n = 50$.



(a) $a = 0.25$ (b) $a = 0.75$

Figure 6: Standard deviation of the convergence of the evaluated algorithms over 30 independent runs for $n = 50$.

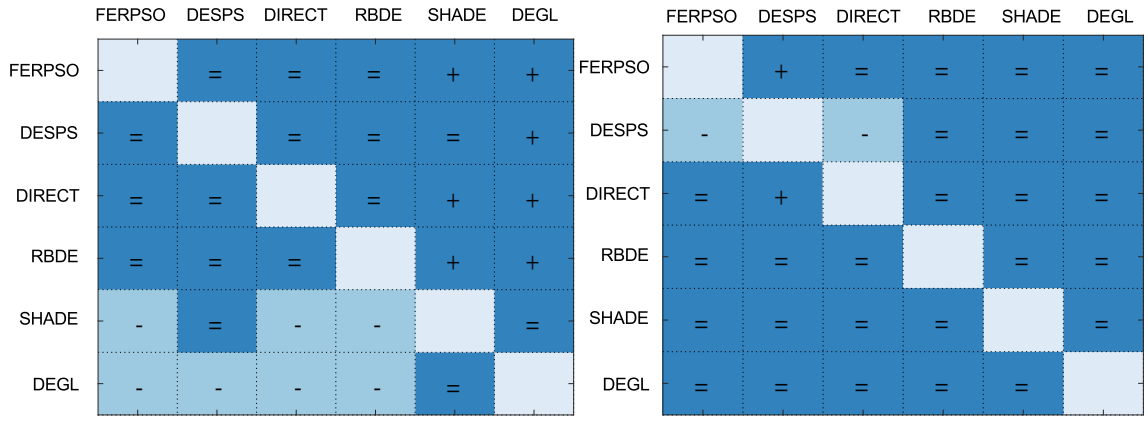
search space and generates combinations by the parallel reduction in the Graphics Processing Unit (GPU) based on the minimal change order. Our computational experiments have shown (1) the feasibility of generating combinations efficiently within a reasonable computable range and (2) the feasibility of using gradient-free optimization algorithms to find combination objects that meet the user-defined aggregation values within a reasonable number of evaluations. Our GPU approach aims to realize the practical efficiency for resource allocation problems, such as agents, space, memory, time, throughput, and capital. We aim to study further combinatorial problems and tailored search strategies to explore the frontiers of the nature-inspired optimization algorithms aided by enumerative encodings. When using numbers as representation, the search space becomes one-dimensional and is amenable to parallelization schemes. We believe our work opens the door to tackle combinatorial problems by GPU-based implementations.

ACKNOWLEDGEMENT

This research was supported by JSPS KAKENHI 20K11998.

REFERENCES

[1] A. F. Myers, "k-out-of-n: G System Reliability with Imperfect Fault Coverage." IEEE Transactions on Reliability, Vol. 56. No. 3 pp. 464-473, 2007.
[2] Y. Tamada, S. Imoto and S. Miyano, "Parallel Algorithm for Learning Optimal Bayesian Network Structure", The Journal of Machine Learning Research, Vol. 12, pp. 2437-2459, 2011.
[3] M. Ventresca, "Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem", Computers & Operations Research, Vol. 39, pp. 2763-2775, 2012.
[4] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, V. Gurvich, K. Makino, "Enumerating spanning and connected subsets in graphs and matroids", J. Oper. Res. Soc. Jpn., 50 (2007), pp. 325–338.
[5] K. Suzuki and M. Yokoo, "Secure combinatorial auctions by dynamic programming with polynomial secret sharing", 6th International Conferene on Financial Cryptography, Lecture Notes in Computer Science, Vol. 2357, pp. 44-56, 2003.
[6] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, A.I. Verkamo, "Fast discovery of association rules", Advances in Knowledge Discovery and Data Mining (1996), pp. 307–328.
[7] V. Parque, T. Miyashita, "Bundling n-Stars in Polygonal Maps", International Conference on Tools with Artificial Intelligence (ICTAI), pp. 358-365, 2017.
[8] V. Parque, T. Miyashita, "Obstacle-Avoiding Euclidean Steiner Trees by n-Star Bundles", International Conference on Tools with Artificial Intelligence (ICTAI), 315-319, 2018.
[9] V. Parque, "On Hybrid Heuristics for Steiner Trees on the Plane with Obstacles", The 21st European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP), pp. 120-135, 2021.



(a) $a = 0.25$ (b) $a = 0.75$

Figure 7: Statistical significance based on the Wilcoxon test at 5% significance level. Symbols with '+/=/-' denote instances where an algorithm in the row is significantly better/similar/worse to an algorithm in the column.

[10] B. J. Chisholm, D. C. Webster, "The development of coatings using combinatorial/high throughput methods: a review of the current status", Journal of Coatings Technology and Research, Vol. 4, No. 1, pp. 1-12, 2007.

[11] T. Imada, S. Ota, H. Nagamochi, T. Akutsu, "Efficient enumeration of stereoisomers of outerplanar chemical graphs using dynamic programming", J. Chem. Inf. Model., 51, pp. 2788–2807, 2011.

[12] T. Eiter, K. Makino, "On computing all abductive explanations from a propositional Horn theory", J. ACM, Vol. 5, No. 24, 2007.

[13] "Why is combinatorial communication rare in the natural world, and why is language an exception to this trend?", Journal of the Royal Society Interface, Vol. 10, No. 88, 2013.

[14] S. Martello, P. Toth, "Knapsack Problems: Algorithms and Computer Implementations", Wiley, Chichester, UK, 1990.

[15] M.R. Garey, D.S. Jonson, "Computers and Intractability: A guide to the Theory of NP-completeness", WH. Freeman & Co, San Francisco, 1979.

[16] A. Lobstein, "The hardness of solving subset sum with preprocessing", IEEE Trans. Inf. Theory 36 (4) pp. 943–946, 1990.

[17] J.C. Lagarias, A.M. Odlyzko, "Solving low density subset sum problems", 24th Annual Symposium on Foundations of Computer Science, pp. 1–10, 1983.

[18] J.H. Ahrens, G. Finke, "Merging and sorting applied to the 0-1 knapsack problem", Operations Research 23, pp. 1099-1109, 1975.

[19] B. Faaland, "Solution of the value-independent knapsack problem by partitioning", Operations Research 21, pp. 332-337, 1973.

[20] D. Pisinger, "An O(nr) algorithm for the subset-sum problem, Report 95/6, DIKU, University of Copenhagen, Denmark, 1995

[21] N. Y. Soma, P. Toth, "An exact algorithm for the subset sum problem", European Journal of Operations Research, Vol. 136, pp. 57-66, 2002.

[22] H. Kellerer, R. Mansini, M. G. Speranza, "Two linear approximation algorithms for the subset-sum problem", European Journal of Operations Research, Vol. 120, pp. 289-296, 2000.

[23] W.L. Chang, et al., "Quantum algorithms of the subset-sum problem on a quantum computer", WASE International Conference on Information Engineering, ICIE, pp. 54-57, 2009.

[24] T. Mine, Y. Murakami, "An implementation of space–time tradeoff method for subset sum problem", Sixth International Conference on Computer Sciences and Convergence Information Technology (ICCIT), pp. 618–621, 2011.

[25] D. Ghosh, N. Chakravarti, "A competitive local search heuristic for the subset sum problem", Comput. Oper. Res. 26, Vol. 3, pp. 271 - 279, 1999.

[26] Shenshen Gu, Rui Cui, "An efficient algorithm for the subset sum problem based on fi nite-time convergent recurrent neural network", Neurocomputing, Vol. 149, pp. 13 - 21, pp. 13-21, 2015.

[27] L. Wan, K. Li, K. Li, "A novel cooperative accelerated parallel two-list algorithm for solving the subset-sum problem on a hybrid CPU–GPU cluster", J. Parallel Distrib. Computation, Vol. 97, pp. 112-123, 2016.

[28] C.A.A. Sanches, N.Y. Soma, H.H. Yanasse, "Parallel time and space upper-bounds for the subset-sum problem", Theoretical Computer Science, Vol. 407, pp. 342-348, 2008.

[29] E. Horowitz, S. Sahni, "Computing partitions with applications to the knapsack problem", Journal of the Association for Computing Machinery 21, pp. 277-292, 1974.

[30] R.L. Wang, "A Genetic Algorithm for Subset Sum Problem", Neurocomputing, pp. 463-468, 2004.

[31] Oberoi, A., Gupta, J.: On the applicability of genetic algorithms in subset sum problem. Int. J. Comput. Appl. 145(9), 37–40 (2016)

[32] H. Wang, Z. Ma, I. Nakayama, "Effectiveness of penalty function in solving the subset sum problem", Proceedings of IEEE International Conference on Evolutionary Computation, pp. 422-425, 1996.

[33] Thada, V., Shrivastava, U.: Solution of subset sum problem using genetic algorithm with rejection of infeasible offspring method. Int. J. Emerg. Technol. Comput. Appl. Sci. 10(3), 259–262 (2014)

[34] Saketh, G. Comparison of Dynamic Programming and Genetic Algorithm Approaches for Solving Subset Sum Problems. In Computational Vision and Bio-Inspired Computing (pp. 472–479). Springer International Publishing, 2020

[35] Rohlfshagen, P., Yao, X. "Dynamic combinatorial optimisation problems: an analysis of the subset sum problem". Soft Comput 15, 1723–1734 (2011). https://doi.org/10.1007/s00500-010-0616-9

[36] I. M Comsa, C. Grosan, S. Yang, Dynamics in the Multi-objective Subset Sum: Analysing the Behavior of Population Based Algorithms, Evolutionary Computation for Dynamic Optimization Problems pp 299-313, 2013

[37] Zhou, J. On the Running Time Analysis of the (1+1) Evolutionary Algorithm for the Subset Sum Problem. In Bio-Inspired Computational Intelligence and Applications (pp. 73–82). Springer Berlin Heidelberg, 2007.

[38] Branke J, Orbayi M, Uyar S The role of representations in dynamic knapsack problem. In: Rothlauf F (ed) EvoWorkshops 2006. Springer, Berlin, pp 764–775

[39] V. Parque, T. Miyashita, ""On k-subset sum using enumerative encoding", IEEE International Symposium on Signal Processing and Information Technology, pp. 81-86, 2016

[40] V. Parque, "Tackling the Subset Sum Problem with Fixed Size using an Integer Representation Scheme", IEEE Congress on Evolutionary Computation (CEC), Krakow, Poland, 2021.

[41] Zhuo Li, Jiannong Cao, Zhongyu Yao, Wengen Li, Yu Yang, and Jia Wang. 2020. Recursive Balanced k-Subset Sum Partition for Rule-constrained Resource Allocation. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM '20). Association for Computing Machinery, New York, NY, USA, 2121–2124.

[42] V. Parque, M. Kobayashi, M. Higashi, "Bijections for the numeric representation of labeled graphs", IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 447-452, 2014.

[43] V. Parque, T. Miyashita, "On succinct representation of directed graphs", IEEE International Conference on Big Data and Smart Computing (BigComp), 199-205, 2017.

[44] V. Parque, T. Miyashita, "On the Numerical Representation of Labeled Graphs with Self-Loops", International Conference on Tools with Artificial Intellignce (ICTAI), pp. 342-349, 2017.

[45] D. E. Knuth, "Generating All Combinations and Partitions", The Art of Computer Programming, Fascicle 3, Addison-Wesley, pp. 5-6.

[46] C. J. Mifsud, "Algorithm 154: combination in lexicographical order", Communications of the ACM, Vol. 6, No. 3, pp. 103, 1963.

[47] P. J. Chase, "Algorithm 382: Combinations of M out of N objects", Communications of the ACM, Vol. 13 No. 6, pp. 368, 1970.

[48] C. N. Liu and D. T. Tang, "Algorithm 452: enumerating combinations of m out of n objects", Communications of the ACM, Vol. 16, No. 8, pp. 485, 1973.
[49] S. G. Akl, "A Comparison of Combination Generation Methods", ACM Transactions on Mathematical Software, Vol. 7, No. 1, p.42-45, 1981.
[50] C. Martinez, X. Molinero, "An experimental study of unranking algorithms", C.C. Ribeiro, S.L. Martins (Eds.), Experimental and efficient algorithms. Lecture notes in computer science, vol. 3059 (2004), pp. 326–340.
[51] F. Ruskey and A. Williams, "The coolest way to generate combinations", Discrete Mathematics, Vol. 309, pp. 5305-5320, 2009.
[52] L. Xiang and K. Ushijima, "On O(1) time algorithms for combinatorial generation", The Computer Journal, Vol. 44, No. 4, 2001.
[53] P. Flajolet, P. Zimmerman, B. Van Cutsem, "A calculus for the random generation of combinatorial structures", Theoretical Computer Science, 132 (1) (1994), pp. 1–35
[54] D. E. Knuth, "The Art of Computing Programming", Vol. 2 Seminumerical Algorithms, Addison Wesley, Reading, Mass., 1968.
[55] J. Kurtzberg, "Algorithm 94: Combination", Communications of the ACM, Vol. 5, No. 6, pp. 344, 1962.
[56] Xiaodong Li. A multimodal particle swarm optimizer based on fitness Euclidean-distance ratio. In Proceedings of the 9th annual conference on Genetic and evolutionary computation. Association for Computing Machinery, New York, NY, USA, 78–85. DOI:https://doi.org/10.1145/1276958.1276970
[57] S. Guo, C. Yang, P. Hsu and J. S. -. Tsai, "Improving Differential Evolution With a Successful-Parent-Selecting Framework," in IEEE Transactions on Evolutionary Computation, vol. 19, no. 5, pp. 717-730, Oct. 2015, doi: 10.1109/TEVC.2014.2375933.
[58] Jones D.R. (2001) Direct Global Optimization Algorithm. In: Floudas C.A., Pardalos P.M. (eds) Encyclopedia of Optimization. Springer, Boston, MA. https://doi.org/10.1007/0-306-48332-7_93
[59] Jones, D.R., Martins, J.R.R.A. The DIRECT algorithm: 25 years Later. J Glob Optim 79, 521–566 (2021). https://doi.org/10.1007/s10898-020-00952-6
[60] Andrew M. Sutton, Monte Lunacek, and L. Darrell Whitley. 2007. Differential evolution and non-separability: using selective pressure to focus search. In Proceedings of the 9th annual conference on Genetic and evolutionary computation, Association for Computing Machinery, New York, NY, USA, 1428–1435. DOI:https://doi.org/10.1145/1276958.1277221
[61] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for Differential Evolution," 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 2013, pp. 71-78, doi: 10.1109/CEC.2013.6557555.
[62] S. Das, A. Abraham, U. K. Chakraborty, A. Konar, "Differential Evolution using a Neighborhood-Based Mutation Operator", IEEE Transactions on Evolutionary Computation, Vol. 13, No. 3, pp. 526-553, 2009.