2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

GEAS: A GA-ES-Mixed Algorithm for Parameterized Optimization Problems - Using CLS Problem as an Example

Xing Zhou, Wei Peng and Bo Yang

***Abstract*— Parameterized optimization problems (POPs) belong to a class of NP problems which are hard to be tackled by traditional methods. However, the relationship of the parameters (usually represented as k) makes a POP different from ordinary NP-complete problem in designing algorithms. In this paper, GEAS, an evolutionary computing algorithm (also can be seen as a framework) to solve POPs is proposed. This algorithm organically unifies genetic algorithm (GA) framework and the idea of evolutionary strategy (ES). It can maintain diversity while with a small population and has an intrinsic parallelism property:each individual in the population can solve a same problem that only has a different parameter. GEAS is delicately tested on an NP-complete problem, the *Critical Link Set Problem*. Experiment results show that GEAS can converge much faster and obtain more precise solution than GA which uses the same genetic operators.**

I. Introduction

PARAMETERIZED optimization problems are easy to be found in daily life. For example, the knapsack problem that one should choose *at most* k objects from n objects, put them into a bag to maximize the total profit, while the total volume of these objects can't exceed the volume of the bag. In this problem the parameter is k. This type of POP can be stated as below:

$$\begin{array}{ll} \text{maximize/minimize} & M(x) \\ \text{subject to} & f(x) = g(k) \\ & x \in \Omega \end{array} \quad (1)$$

where M is the objective function for the problem, f and g are functions. Line 3 in equation (1) gives the variable space of the problem. However, Line 2 means that only those points which satisfy the equality should be considered, that is to say the variable space is further reduced by Line 2. Line 2 contains the parameter, so equation (1) defines a POP with parameter equaling to k.

Taking the k-objects knapsack problem as an example to illustrate equation (1): x can be a binary string with length n. M is the profit function on x. f could be the hamming weight function, i.e., $f(x)$ counts the number of bit 1s in x. g is the identity function in this example and $\Omega = \{0,1\}^n$.

The critical link set problem (CLS) emerging from complexity network research is a POP too. CLS with parameter k, asks how much destruction can be caused at most, if we delete k links (edges) in a network. These k edges that destruct most consist of the set named critical link (edge) set. A lot of other problems in complexity network research resemble the CLS problem, such as the *critical node set problem*[9], the *maximum prohibited flow* in complex networks[10]. So, they are also parameterized optimization problems.

Almost all these parameterized optimization problems are NP-complete [1] [2]. Hence, it's unlikely that there is polynomial time exact algorithms to solve these problems.Branch-and-bound search is one approach to get the optimal solution. But it's hard to get the optimal solution efficiently for the combinatorial explosion and gigantic enumeration.

Sometimes, dynamic programming techniques can generate a pseudo-polynomial time exact algorithm for a NP-complete problem, like the ordinary knapsack problem. But dynamic programming requires a problem two features: duplicate subproblems and optimal substructure [3]. It's difficult to design a dynamic programming algorithm to the mentioned POP problems, because (1) What is the subproblem? how can we divide current problem to similar subproblems? (2) These optimization problems seems have no optimal substructure feature, too. For example, optimal critical link set of size $k-1$ don't have to be a part of CLS of size k.

For their practical importance, a lot of efforts have been devoted to these problems, trying to find a good way to get the optimal solution, or even near-optimal solutions. Approximation algorithms is a choice if we cannot get the exact optimal solution within a bearable time. We can accept an approximation algorithm if its result is not far from the optimal one, say at most two times bigger (smaller) than that of exact algorithms. The LP relaxing is one approach to produce approximation solutions: at first formulate a optimization problem as an integer linear program (ILP); then relax the ILP to linear programm (LP) by cancelling the integer restriction of the variables; then, solve the LP model by Simplex[4] method to get real number solution; finally round the real number solution to integer solution according to some appropriate strategy. It seems a good way to get near-optimal solutions, but it's difficult for the solution to remain good quality after the rounding stage, and unappropriate rounding strategy may bring wide gap solution or even invalid solution.

What's more, theoretically been proven, some of the problems are hard to be approximated within a constant times' gap to the optimal solution. For example, the CLS problem has been proved that it's hard to be approximated within ratio of $\Omega(\frac{n-k}{n^\epsilon})$ in polynomial time, where k is the number of deletions and $\epsilon < log_n 2$.

Xing Zhou, Wei Peng and Bo Yang are with the College of computer, National University of Defense Technology, Changsha, China(email: {zhouxing, wpeng, yangbo}@nudt.edu.cn)

This work was partly funded by the research project of National University of Defense Technology, and National Natural Science Foundation of China under grant No.61070199, 61100223 and 61272010.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

Previous works were mainly focused on single parameterized problem and approximation algorithms were often derived from Linear Programming. Previous works have not utilized the relationship of the parameters in POP. By instinct, POPs with different k (can effect each other. A POP with certain parameter can be greatly effected by the same POP but with other parameters. Thus we can use this properties and approximate these problems using evolutionary computing, solving them on an algorithm (framework) we propose that mixes the framework of GA [5] and the idea of ES [6].

GEAS firstly generates a small population randomly or using problem-specific knowledge. Each individual in the population represents a valid solution to the POP with certain parameter (called "a problem"). There are at least one individual corresponding to the problem with parameter equaling to k. After the initialization, at each generation an individual mates with another one and produce an offspring. An individual can also mutate to evolve. Each individual keeps the best solution for its problem (i.e. for certain parameter) found so far. After the evolutionary procedure, each individual in the population expresses a near-optimal solution to its problem. At last we get an approximate solution to the problem with parameter k, and a bunch of solutions to the same problem but with other parameters (in some circumstance, we are seeking these solutions, too. so this algorithm will especially bring us loads of convenience and save us a lot of time in that case). This algorithm has the following features.

- This algorithm can get more accurate solutions than similar GA. This algorithm is less likely to be trapped to local optimality than GA. Its population constituent, crossover and mutation pattern can help escape from local optimality.
- Intrinsic parallelism property. The algorithm betters off every individual without bias, so each individual will be an good approximate solution to the corresponding parameterized problem at the end of evolution. These problems are slightly different from each other only in parameter.
- It has a small population, and each individual is an elite since the beginning of the evolution. The superiority gene for each problem is kept in the corresponding elite individual. The diversity for each problem with certain parameter is reserved in the corresponding individual and in other individuals, too.
- The running time on this algorithm is less than that on the basic genetic algorithm framework owing to its small niche. Because of the well maintained superiority and diversity, the framework converges quickly.
- The selection procedure involves no fitness function.

We choose the CLS problem as an experimental subject in this paper. The convergency, accuracy and time difference between the basic GA framework and GEAS are systematically compared/analysed.

The paper is organized as follow. Section II is the preliminary knowledge of CLS, introduction to basic GA framework and ES. Section III describes GEAS. Section IV is the algorithmic implementation for CLS on both GEAS and GA, including the design of crossover operator and the mutation operators. Section V shows the experimental results and the analysis. Section VI concludes this paper.

II. PRELIMINARY

A. critical link set problem

The critical link set problem is to find a certain number of links whose removal will destruct the connectivity of a network to the maximal extent. It is a fundamental problem in the evaluation of the vulnerability or robustness of a network because a network's performance highly depends on its topology. In a typical attacking scenario, the attacker first figures out the weak part of a network, e.g., some key communication links. The attacker then tries to disrupt the links or to bring them down. The connectivity or performance of the targeted network will be degraded once the links fail. The critical link set problem is defined as: find at most k links in a network whose removal will degrade the network performance to the maximum extent.

A lot of metrics have been applied to measure the topological performance of a network [7]. To well measure the vulnerability of a network, like the communication network, the pairwise connectivity [8] –the total number of connected node pairs–is used. By its definition, we know, the pairwise connectivity of a connected component with n nodes is C_n^2. And the total pairwise connectivity of a graph is the summation of its components'. For convenience, we denote critical link set problem as CLP, pairwise connectivity as PC. Figure 1 shows a simple instance of a network. For CLS problem with parameter $k = 1$ in this network, the critical link set=$\{(0,1)\}$, and the PC after deletion is 3. If $k = 2$, the critical link set=$\{(0,2),(0,3)\}$, and the PC is 2.

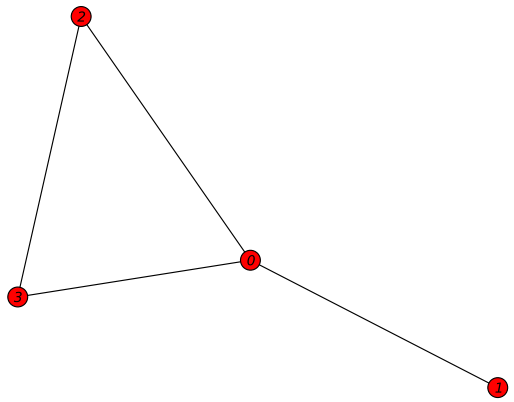

Fig. 1. a simple network

A graph (or component) G is $k-edge-connected$ if the removal of any $k-1$ edges will not make G disconnected, but there exists a removal of k edges that does. Sometimes, it's

called $k-connected$ for short. $k-connected-components$ means a set of nodes together with the edges linking them. In Fig. 1, edge (0,1) and node 0, node 1 is $1-connected$, while edge (0,2), (0,3), (2,3) and node 0,2,3 is $2-connected$. In the following text, a $k-connected-components$ only means the edge set because the edge set determines the nodes that relate to it. we call $2-connected$ is "higher" connected than $1-connected$ for convenience in following. The $k-connected$ conception will be used in initialization part of GEAS.

A bridge edge, also called a bridge, is an edge whose deletion will disconnect a connected component (graph). In Fig. 1, edge (0,1) is an bridge edge. If the parameter of the CLS problem we are solving is 1, then a bridge will be a good solution. If there exists many bridges, we can choose the best one by comparing the remaining PC after the bridge is removed. We will use the concept of bridge in the mutation operator ($Descendor$) in later.

B. genetic algorithm

A genetic algorithm (GA) [5] is a search heuristic that imitate the process of natural selection. This heuristic is often used to generate satisfiable solution to optimization and search problems. In a GA, a population of candidate solutions (individuals) are generated and evolved toward better direction. Each individual has a set of chromosomes or genotypes that can be mutated and altered. The chromosome representation is called a code scheme. It is a binary string of 0s and 1s at the early history, but other encoding are possible now.

The evolution stage is a iterative process, with each iteration called a *generation*. In each generation, the fitness (goodness) of each individual is calculated. The fitness is a number indicating our content degree toward a individual. The larger the fitness, the more content we are. The higher fit individuals are selected out to reproduce new individuals (recombination or mutation) to form the new population. The new population is then used in the next iteration. The algorithm ends when satisfactory solution has been found or it reached the maximum generation.

C. evolutionary strategies

Evolutionary strategies (evolution strategies, ES) use natural problem-dependent and mainly mutation and selection as search operators. These operators are applied in a loop as the GA is. The selection in ES is deterministic and only based on the fitness rankings, not on the real fitness value. The simplest ES operates on the current point (parent) is to mutate it to a mutant. Only if the mutant's fitness is better than the parent one, it replace the parent. Otherwise, the mutant is abandoned. This is a $(1+1)$-ES. Moreover, if λ mutants can be generated and compete with the parent, this is $(\lambda+1)$-ES. There are also many other strategies.

As the development of evolutionary computing, the difference between GA and ES is smaller and smaller.

III. GEAS Algorithm

The inspiration of GEAS comes from this observation: although a POP with parameter k has no direct impact on problem with larger parameters, and is not directly impacted by problem with smaller parameters, but these problems are implicitly affected by each other via parameters, so we can design evolutionary algorithms based on parameters.

We now give the algorithm (framework) GEAS in Algorithm 1.

Algorithm 1 GEAS

Input:
essential input: M, Ω, f, g, k as mentioned in equation (1);
non-essential input: the maximum generation *MAXGEN*, the population size *pop*;
Output:
the best x in Ω and $M(x)$;

```
1: Generate pop individuals randomly or using problem-specific knowledge, there should exist at least one individual λ such that f(λ) = g(k);
2: old=this population;
3: for each i ∈ old do
4:    fitness_i = compute_fitness(i)
5: end for
6: while process not stop do
7:    for each i ∈ old do
8:       Select a mate m_i for i randomly or intentionally;
9:       if crossover condition fits then
10:         crossover i and m_i to get new_i;
11:      else
12:         copy i to new_i;
13:      end if
14:      if mutation condition fits then
15:         mutate on new_i;
16:      end if
17:   end for
18:   for each new_i do
19:      if (temp = compute_fitness(new_i)) > fitness_i then
20:         replace i ∈ old with new_i;
            fitness_i = temp;
21:      end if
22:   end for
23: end while
24: Further improve the solution and output it.
```

As can be seen, it has a skeleton of basic GA framework. But

(1) the initialization part (Line 1) decides the feature of GEAS and is quite different from GA. In GA, its initialization will only generate individuals whose f value equals to $g(k)$ for optimization problem with parameter k. But in GEAS, we generate individuals that are invalid and appear *"irrelevant"* to optimization problems with parameter k. Line 1 is critical to GEAS, because it utilizes the properties of optimization

problems: individual λ such that $f(\lambda) = g(k)$ can be positively affected by other individuals whose f value equals to, say, $g(k-1)$,$g(k+3)$. The initialization part produces the constituent of the population, and this kind of constituent makes GEAS quite different from GA.

(2) selection in Line 8 is not fitness based. By "intentionally", we mean "strategically randomly ". The fitness computed in Line 4 is only used for comparison in Line 19, but not in selection. Though selection in Line 8 is not fitness based, the principle that select high quality parents to reproduce high potential offspring is the same as selection in basic GA.

(3) the substitution part (Line 18 to Line 22) resembles ES. Each individual represents a solution corresponding to a problem with certain parameter. For each problem, we only reserve the better individual between the parent and the son. It is like a (1+1)-ES.

More explanation on GEAS:

- The input is classified to two categories. The essential category defines an optimization problem with parameter k. This tells us what problem to solve, so it's necessary. The non-essential input is mainly for controlling the algorithm, such as the size of the population, the arguments for evolution, etc. These arguments are set by users themselves. The population can be very small in GEAS, containing only dozens of individuals.
- the initial population have impact on GEAS. So a problem-specific generating method is suggested. If no relative knowledge to use, randomly generating will be ok.
- $compute\_fitness()$ is a function on individuals. It returns a quantitative measure for a individual. The large the fitness, the more we want the individual.
- Line 10 is the crossover operator. Various operators can be designed according to problems. But at least one of the parent should be i, because i has the superiority gene for the corresponding problem, or else the offspring may lost the advantage gene to the corresponding problem.
- Line 15 is the mutation operator. GEAS is flexible to include elegant mutation operators. In our simulation for CLS, we designed two mutators: $Swappor()$ and $Descendor()$, both will be described later.
- Line 24 is to further improve the final solution. We can improve it using common sense or problem-specific properties. This improvement may not cost us much time, but can bring us some additional profits.

Following we will introduce the detailed implementation of CLS on both GEAS and GA framework that uses the same genetic operators. After that, we will make comparisons.

IV. IMPLEMENTATION OF GEAS AND GA

A. GEAS

CLS has been introduced in section 1. It is to find k edges so that their deletion makes a network's pairwise connectivity (PC) minimize.

1) Encoding scheme: Each individual in the population encodes a possible deletions, so we use a set "x" to encode the deletions. Elements in a set should not repeat, and this is what we are looking to for solving the problem. A set can also easily to count its number of elements–the size of the set. For example $x = \{(0,2),(0,3)\}$ is a 2-deletion to the network in Figure 1. Because the network we considering is undirected, so edge (0,2) is also (2,0). Be careful to ensure (0,2) and (2,0) not appear in x at the same time.

2) Initialization: This part is Line 1 to Line 2.

According to our previous introduction to $k-connected-components$, an edge in low connected components should more be likely to be deleted than in high connected components (but it's not absolute).

So, we first identify each edge, how high connected components it can belong to. We then assign a value to the edge to indicate its probability for being chosen to delete. Thanks to [11], the authors proposed a polynomial time algorithm that can complete such task. That algorithm takes Stoer's minimum cut algorithm on undirected graph [12] as a base. In our implementation, if an edge belongs to $m-connected-components$, its value is $1.0/m$.

Then, to generate an individual for the problem with parameter k, we randomly draw out k edges from the network according to the probability distribution. At least one individual for parameter k must be generated. And we generating other individuals for problems with parameters in range $[k-pop/2, k-1], [k+1, k+pop/2]$, i.e., the whole population incorporates $pop+1$ individuals with continuing parameters and k is in the middle. This population style is good for obtaining diversity to solve CLS problem.

3) Fitness: The fitness is the preciousness of an individual. Since we want to minimize the PC, the fitness can define $fitness = PC_{original} - PC_{remaining}$. The fitness will not be used in the selection process, so we can have different and simple definition, for example we can replace $PC_{original}$ with a reasonably large integer. To count the $PC_{remaining}$ after the deletion of edges of a individual, we only need to have a traversal (DFS, BFS, etc.) to the remaining graph.

4) Selection: This is in Line 8. If we do not want to bother finding a highly suitable mate for i, we can randomly choose an individual as m_i. m_i should not be too close to i, especially in the later stage of evolution, because too close means too similar, which may cause local optimality.

5) Crossover operator: Suppose we have two individuals a, b, their corresponding problems are with parameters k_a and k_b. We want to generate a child c with parameter k_b. Firstly we let $c = a \cap b$, and the size of c will not be greater than b because of the property of intersection set. If c require additional edges to become a set of size k_b, we randomly choose extra edges from $a \cup b - a \cap b$. This operator well reserves the superiority of the gene for problem with parameter k_b, and also generates diversity.

6) Mutation operator: One of our mutation operator is the Swappor(). Let x be a deletions. Let e be an edge in x, and e' be a neighbour of e but not in x. A neighbour edge of e means this edge has a common endpoint with e. If

containing e' in the critical link set is better than containing e we can replace e with e'. The Swappor is a recursive process as below:

Algorithm 2 Swappor Operator

```
Swappor(individual x){
for each edge e ∈ x do
  if compute_fitness((x − e) ∪ e′) > compute_fitness(x) then
    x = (x − e) ∪ e′ ;
    Swappor(x);
    break;
  end if
end for
}
```

Another mutation operator is called $Descendor()$. In our implementation on GEAS, the individuals represent problems with parameter varies from $k - pop/2$ to $k + pop/2$. For any two adjacent individuals, the latter one should have less or equal PC than the former one, because one more edge is deleted in the latter one. That's to say all the PCs of these individuals form a descending (non-ascending) order sequence. When the mutation operator $Descendor()$ operates, it scans the PC sequence of the individuals from head to tail. where the descending order is broken, $Descendor$ repairs the order as this: Replace the latter individual with a copy of the former one. Then in order to be valid, the latter one should delete one more edge. If there exist bridges in the remaining graph for the latter one, we can delete any bridge in it. Tarjan's algorithm [13] [14] is an effective way to find all bridges in a graph. In fact, we can terminate Tarjan's algorithm in an early stage once a bridge has been found. Certainly, if you find out all the bridges and choose the optimal bridge (see Preliminary), it will be better. If there exists no bridge edges, we randomly choose an edge from the remaining graph to delete.

The Descendor() can't apply to GA, because all the individuals in GA are for problem with same parameter k, rather like GEAS for parameters in a range.

B. GA

1) Encoding scheme: Encode a deletion of k edges in a set. The same as GEAS's.

2) Initialization: Generating $pop + 1$ individuals, but individuals are only valid to the problem with parameter k. For each individual, we use the probability distribution as GEAS's to draw out k edges to form a individual.

3) Fitness: The selection on GA is fitness-based. But our previous definition of fitness for GEAS can still work here, i.e. $fitness = PC_{original} - PC_{remaining}$.

4) Selection: Use the ordinary wheel method to choose a mate.

5) Crossover operator: As GEAS.

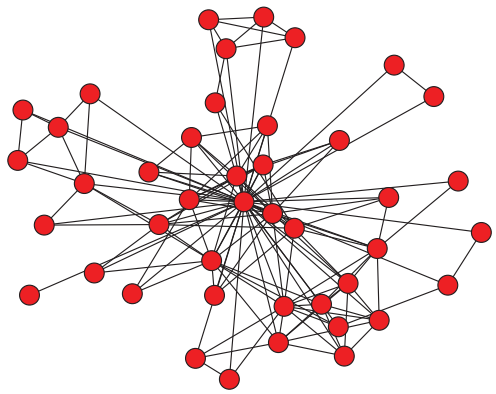

Fig. 2. the terrorists' network

6) Mutation operator: This algorithm has no descending order property, so $Descendor$ can't be applied. But we can still use $Swappor$. To be fair during the comparison between the two algorithms, we do not use Descendor. However, when GEAS compares to GEAS itself, Descendor can be used.

7) Elitism: The ES ingredient in GEAS has elitism. To fairly compare GEAS, we use elitism in basic GA too. We choose the best individual(s) in old generation to replace the worst individual(s) in new generation if the old one(s) is better than the young one(s).

V. Results and Analysis

Our experiments are carried on the 9•11 terrorists network, which is compiled by Krebs. The network has 43 nodes each presenting a terrorist and 139 edges, the contacts between terrorists. The network is shown as Fig. 2.

GEAS has the skeleton of GA. The genetic operators in both GEAS and GA need same time cost. So the complexity of the two algorithms are close, except that GEAS's selection has a constant-time low complexity, while GA has an $O(pop)$ complexity. Overall, GEAS has a complexity one order less than GA.

Fig. 3 shows the comparison between GEAS and GA. The maximum generation are both 1000 and population size = 100. Actually, the population of GEAS can be smaller than GA, but for clarity reason, we generate two population of same size. For fair comparison, we did not use $Descendor$ in GEAS but use elitism in both. The solutions are nearly the same at two ends, that's because when k is too large or too small, both algorithms can find optimal solution for k, so we have not show these results in figures. The results show GEAS has prominent advantages over GA.

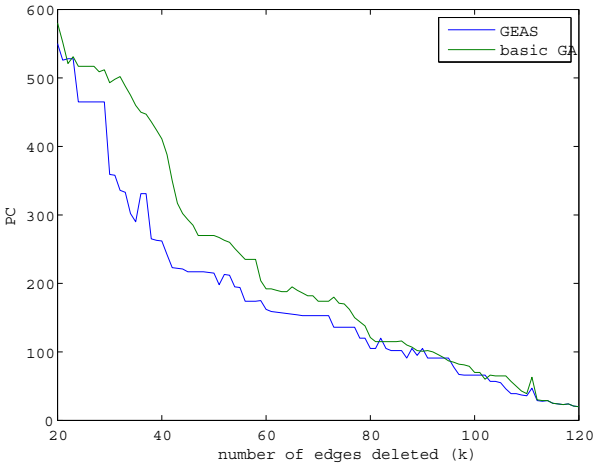

Fig. 3. the final result of basic GA and GEAS

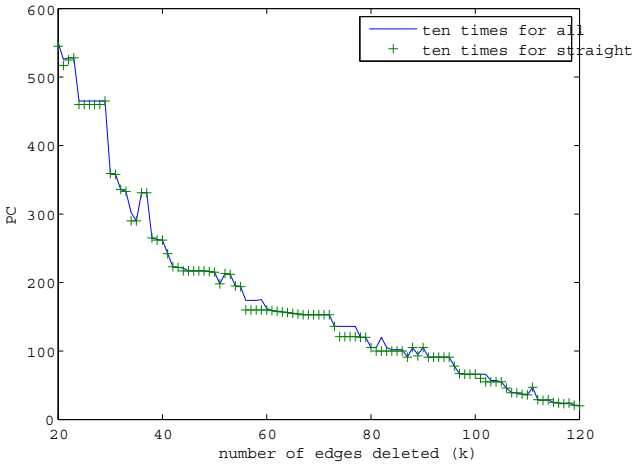

Fig. 4. different running style

Without special declaration, the maximum generation is set to be 1000 and population size is set to be 100 in following context.

In order to get a near-optimal solution for problem with parameter k, we ran the program ten times and choose the minimum solution. We experimented two running style of GEAS. The first one, called "ten times for each": we can run ten times for each parameter k, finds out the minimum PC that has got; then, we change to $k+1$ (i.e., let $k+1$ be in the middle of the population), run the program ten times, and get the minimum PC... Another one, called "ten times for straight": as mentioned before, the algorithm of GEAS optimizes all the individuals at the same time (the parallelism property), so after one running pass, we get the minimum PC for all the parameters in range. We run this ten times, and find the minimum PC for every parameter k. The former style is more troublesome the latter.

Fig. 4 shows result of the two running style. We can see that there is only slight difference between them. Since there is not too much difference, we will use the "ten times for straight" style for its convenience in the following.

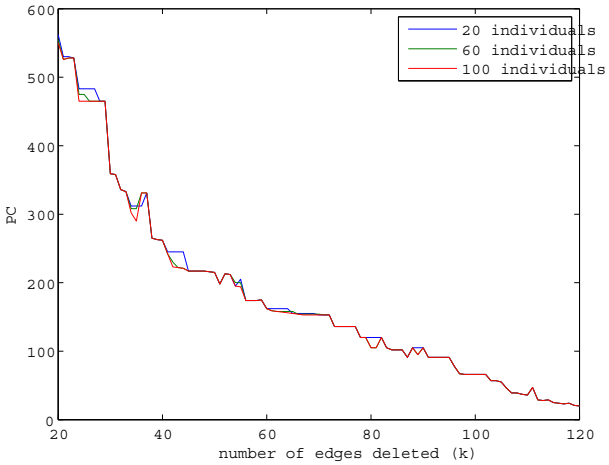

Fig. 5. the effect of population size

Fig. 5 researches the effect of *pop* . We tested *pop*=20, 60, and 100 of "ten times for straight" style. It can be seen that only in particular point that the solutions differ. So dozens of individuals is sufficient for GEAS, this is the reason why GEAS is said to have a small population.

We also simulated how fast convergent the program of GEAS can be. We printed out the temporary solution during the evolution, in Fig. 6. The data are printed after initialization, 10% to 50% generation evolved and the final solution. After 10% generations are evolved, the solutions are very close to the final solutions. Fast convergency is different from early-ripe, because the 10% generation's solution is almost equal to final solutions, and final solutions are better than GA's.

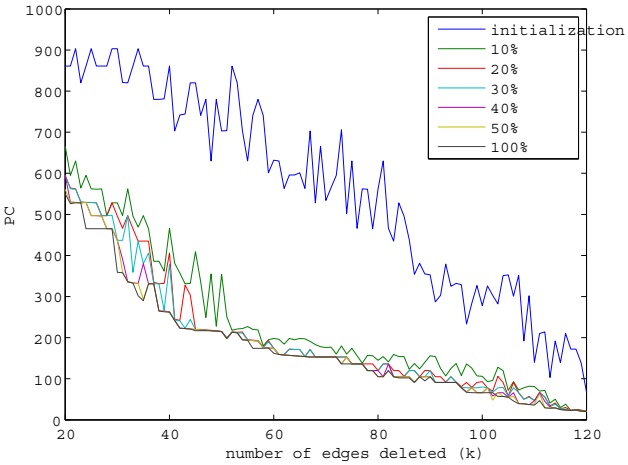

Fig. 6. the solutions during evolution

Fig. 7 shows the results of GEAS with and without mutation operator $Descendor$. We can see, the $Descendor$ wipes off the mountain peaks and improves the solutions.

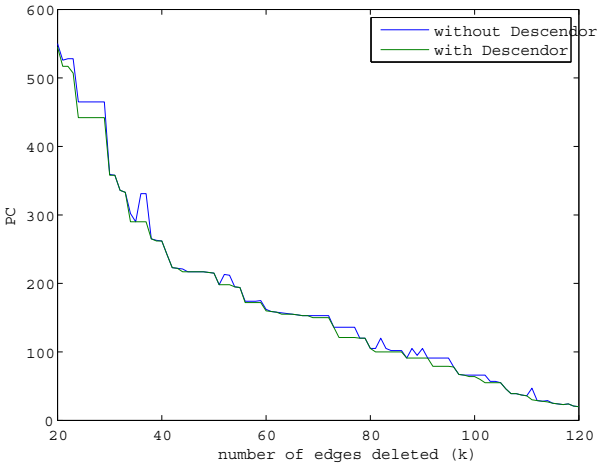

Fig. 7. effect of *Descendor*

VI. Conclusion

In this paper, a new algorithm (framework) that has the features of GA and ES has been proposed. This algorithm is an effective arms to deal with a class of parameterized optimization problems. By systematically testing its convergency, speed and accuracy, it outperforms GA with same genetic operators. More over, it can optimize a lot of similar problems but with other parameters concurrently.

References

[1] M. Cesati.*Compendium of parameterized problems.* Dept. Computer Science, Systems, and Industrial Eng., University of Rome Tor Vergata, 2006.

[2] Y. Shen, N. P. Nguyen, Y. Xuan, et al. *On the discovery of critical links and nodes for assessing network vulnerability.* 2012.

[3] C. E. Leiserson, R. L. Rivest, C. Stein.*Introduction to algorithms.* The MIT press, 2001.

[4] C. W. Churchman, R. L. Ackoff, E. L. Arnoff. *Introduction to operations research.* 1957.

[5] D. E. Goldberg, J. H. Holland. "Genetic algorithms and machine learning". Machine learning, 3(2): 95-99, 1988.

[6] I. Rechenberg. *Evolutionsstrategie–Optimierung technisher Systeme nach Prinzipien der biologischen Evolution.* 1973.

[7] L. D. F. Costa, F. A. Rodrigues, G. Travieso,et al. "Characterization of complex networks: A survey of measurements. " Advances in Physics, 56(1), 167-242, 2007.

[8] T. N. Dinh, Y. Xuan, M. T. Thai, et al. "On new approaches of assessing network vulnerability: hardness and approximation." *Networking, IEEE/ACM Transactions on*, 20(2), 609-619, 2012.

[9] Y Shen., T. N. Dinh, M. T. Thai. "Adaptive algorithms for detecting critical links and nodes in dynamic networks"//MILITARY COMMUNICATIONS CONFERENCE, 2012-MILCOM 2012. IEEE, pp 1-6, 2012.

[10] T. C. Matisziw, T. H. Grubesic, J. Guo. "Robustness elasticity in complex networks". Plos one, 7(7): e39788, 2012.

[11] R. Zhou, C. Liu, J.X. Yu, et al, "Finding Maximal k-Edge-Connected Subgraphs from a Large Graph." In: EDBT 2012, Berlin, Germany, 2012.

[12] M. Stoer, F. Wagner, "A Simple Min-cut Algorithm." J. ACM, 44(4), 585–591, 1997.

[13] R. Tarjan, "Depth-Frist Search and Linear Graph Algorithms." SIAM J. Comput., 1(2), 146–160, 1972.

[14] J. Hopcroft, R. Tarjan, "Efficient Algorithms for Graph Manipulation. " Communications of the ACM, 16(6),372–378, 1973.

[15] V. E. Krebs, "Uncloaking terrorist networks. First Monday". 7(4), 2002.