MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Evolutionary optimization of intrusion detection system in wireless sensor networks

DIPLOMA THESIS

Adam Saleh

Brno, spring 2008

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Adam Saleh

Advisor: Andryi Stetsko, Ph.D.

Acknowledgement

I would like to thank my supervisor ...

Abstract

Wireless Sensor networks are an interesting topic for security research because of their possible real world applications, their distributed nature and the constrains on their hardware. For example, setting up intrusion detection system on network of nodes so that it satisfies constrains on accuracy and hardware can be a timeconsuming. We believe that this process could be greatly improved by using genetic algorithms. Purpouse of this work is to provide a reasonably reusable framework for optimizations with evolutionary heuristics and calibrate it for optimizing of Intrusion Detection System on Wireless Sensor Network.

Keywords

Wsn,ids,spea2,nsgaii

Contents

1 Introduction

2 Evolutionary Optimization of Wireles Sensor Networks

Optimizing configuration of a wireless sensor network can be a time-consuming task. Nodes of the network are constrained devices, so tradeoffs between battery life, network, memory and processor usage have to be considered. Because it is difficult to to predict how will different configuration settings change the performance of the network, it is advisable to measure the impact of changing configuration with WSN simulation software. Unfortunately, simulation of such configuration can take anywhere between several minutes to several hours of processor time, rendering usage of simple exhaustive search on all configurations impractical for most WSN calibration purposes.

According to (b Talbi), optimization problems with problem space that is hard to characterize and expensive to search are good fit for heuristic approaches.

[copy figure 1.9 from talbi]

Definition 1 (Optimization problem)**.** *We define optimization problem as a triplet (X, Z, f), where $X \subseteq \mathbb{R}^n$ is the decision space, $Z \subseteq \mathbb{R}^m$ is the objective space and $f : X \to Z$ is the evaluation function. Without loss of generality it is assumed, that we want to minimize objective.*

Because we focused on multi-objective optimization, we used definition of optimization problem from (c IBEA), mainly to maintain constistency in other definitions in later chapters. This defintions allows the evaluation function f to output incomparable objective vectors, which is a key concept for multi-criteria optimization. Optimization problem can be easily made compatible with single-criteria algorithms by defining cost-function $c : Z \to \mathbb{R}$, that creates an agregatted metric from the objective vector.

2.1 Basic principles of metaheuristics

Most of the heuristics used are variants on search through problem space, with different approaches to avoid undesirable local optima.

There are two basic principles that are used when designing a metaheuristic:

2.1.1 Diversification:

an algorithm should explore the search space.

Random search can be considered an extreme application of difersification, where in every round we generate next solution randomly without using any information from previous (good) solutions.

Less extreme example is the family of population based heuristics, where in each iteration we are trying to improve a set of candidates, called a population.

2.1.2 Intensification:

an algorithm should exploit information from the best solutions found.

Local search can be considered an extreme application of intensification, where next instance to evaluate is allways selected deterministicaly, based on the previous solution.

Local search and its variants belong to a family of single-solution based heuristics, where in each iteration we are basing our search of the problem-space of single solution. (b talbi)

Because of their nature, heuristic approaches are hard to evaluate and reason about, therefore they are often considered to be a method of last resort. (b Talbi) warns against using heuristics laizily on problems where more deterministic approaches (such as using a SAT-solver or linear programming) would be more efficient. Heuristic approaches are usually considered whe we need to solve our problem in order of magnitude faster than possible using conventional approaches, while sacrificing guarantees on optimality.

In our case, we are constrained by number of configuration evaluations we are able to perform in a given timeframe. We are focusing on evolutionary heuristics, because they seem to have reasonable performance with regard of optimizing WSN (b Stetsko 2011). These are modeled natural proces of evolution of species. It is a family os stochastic, population-based heuristics (as opposed to deterministic, single-solution based).

2.2 Single objective evolutionary algorithms

Basic principles behind evolutionary algorithms can be shown on example of simple evolutionary algorithm.

Evolutionary algorithms are based of the fact, that natural proces of evolution can be considered an algorithm solving an optimization problem of adapting a species to its environment.

2.2.1 Template of an evolutionary algorithm. (b Talbi)

Step 0: Initialization: Generate an initial population of individuals

Step 1: Fitness assigment: calculate the fittness values of individuals in population

Step 2: Enviromental selection: select the best individuals of a population to "survive" to the next generation

Step 3: Termination: check if termination criterium holds (usually based on number of generations), if it does, return the result and stop.

Step 4: Variation: apply mutation and crossover parameters on the surviving individuals, increment the generation counter and proceed with step 1.

If we continue with this metaphor, we can find these paralels:

- Population of individuals is a set of configurations. Their environment is the optimization problem.
- We decide surviving individuals based on their evaluation function score.
- From surviving individuals we generate next generation, either by mutation or crossover.
- Mutation of individual usualy consists of randomly selecting some configuration from its neghbourhood as its ofspring. This step provides intensification for evolution heuristic.

- Crosover usualy consists of creating an offspring by permutation of two individuals. This step provides diversification for evolution heuristic.

In algorithms, there are variations on this template, for example, instead of first selecting the best and then generating the rest of population, *variation* can precede *enviromental selection*, and enviromental selection then trims the population to the confines of preset population limit.

In our framework selection and replacement steps are part of the evolutionary algorithm, while crossover and mutation functions are to be calibrated on case by case basis.

2.3 Multiobjective evolution algorithms

Problem that we are trying to optimize is unfortunately illsuited for single objective evolution. Objectives such as memory consumption and IDS accuracy are ortoghonal, therefore it is hard to determine single criteria that would satisfactory cover both of them. A usual solution is to provide some sort of weighted average. In this case,because output of our algorithm would be only a single solution, we need to know emphasis on different criteria before we run our optimization.

Better approach is to recognize multi-objective nature of our problem.

Multi objective algorithms on the other hand are based on idea of Pareto optimality and domination. We say that solution A dominates solution B, if A has no objective "worse" than B and at least one objective "better". We say that A is pareto-optimal, if there is no other solution that would dominate A. This means that no objective of A can be improved without deterioration of another objective. Set of all non-dominated solutions is then called Pareto front.

Subset of solutions where no solution from the superset dominate the one in the set is called Pareto optimal set.

We will call a subset that contains only solutions that don't dominate each other a Pareto aproximation.

Definition 2. *Pareto dominance An objective vector* $u = (u_1, \cdots, u_n)$ *is said to dominate* $v = (v_1, \cdots, v_n)$ *(denoted by* $u \prec v$*) if and only if no*

component of v is smaller[1] *than the coresponding component of u and at least one component of u is strictly smaller:*

$$\forall i \in \langle 1, n \rangle : u_i \leq v_i \wedge \exists i \in \langle 1, n \rangle : u_i < v_i$$

Definition 3. *Pareto optimality A solution $x \in S$ is Pareto optimal if for every $x' \in S$, $F(x')$ does not dominate $F(x)$, that is*

$$\forall x' \in S, F(x) \not\prec F(x')$$

Definition 4. *Pareto optimal set For given MOP(F,S), the pareto optimal set is defined as $\mathcal{P}* = \S \in \mathcal{S} |\ \nexists \S' \in \mathcal{S}, \mathcal{F}(\S') \prec \mathcal{F}(\S)$*

Definition 5. *Pareto front For given MOP(F,S), the pareto optimal set is defined as $\mathcal{P}* = \S \in \mathcal{S} |\ \nexists \S' \in \mathcal{S}, \mathcal{F}(\S') \prec \mathcal{F}(\S)$*

Definition 6. *Pareto aproximation For given MOP(F,S), we call set $A \subset S$ a pareto aproximation if $\forall x \in A\ \nexists x' \in A, F(x') \prec F(x)$*

Multiobjective heuristics then try to obtain best aproximation of Pareto front. To gauge how good an approximation of Pareto front is, we usualy use two criteria, first to measure convergence to Pareto optimal front and second to measure diversity in found pareto set.

[image about approximations]

Calibration of multi objective evolution algorithms is harder than their single criteria counter-parts, mainly because it is hard to reason about the convergence of output without having the real Pareto optimal front in the first place. Pareto optimal sets often aren't directly comparable. TODO We will discuss calibration in greater detail in later chapters.

2.3.1 NsgaII

Step 0: Initialization: Generate an initial population P_0

Step 1: Fitness assigment: calculate the fittness values of individuals P_0

Step 2: Variation: Generate temporary population P'_t by applying crossover and mutation operators on P_t

1. we assume minimization

Step 3: Fitness assigment: calculate the fitness of P'_t

Step 4: Nondominated sort: sort the $P_t \cup P'_t$, first by domination rank, second by crowding metric

Step 5: Enviromental selection: create P_{t+1} as the first N individuals of the sorted $P_t \cup P'_t$

Step 6: Termination: If $t \geq T$, or other stopping criterion is satisfied, return nondominated individuals from P_{t+1}, else increment t and continue with step 2.

We can separate step 4, the nondominated sort into two phases:

Non-dominated sorting

Every individual is assigned a rank based on order of nondominated front it belongs to. The first front is the Pareto front of the population:

$$\mathbb{F}_{1} = x \in P| \nexists x' \in P, x' \prec x$$

Other can be defined recursively as the paretofront of the remaining population not containing previous fronts $R_i = P - \bigcup_{1 \geq j < i} F_j$:

$$\mathbb{F}_{2} = x \in R_i| \nexists x' \in R_i, x' \prec x$$

Crowding distance sorting

Individuals in the same rank are then sorted by their crowding distance. First crowding distance per objective of each individual is calculated.Agregated crowding distance of the individual is then the sum through all the objectives.

Let $\mathbb{P}_{\succcurlyeq}$ be a nondominated set ordered by objective m, and $\mathbb{P}_{\succcurlyeq}[i]$ the value of objective m of individual i, then we can recursivaley. define individuals crowding distance C_i:

$$C_i = \sum_m C_{i_m}$$

, where

$$C_{i_m} = \frac{\mathbb{P}_{\geqslant}[i+1] - \mathbb{P}_{\geqslant}[i-1]}{f_m^{max} - f_m^{min}}$$

It has to be noted, that for agregation to work, each C_{i_m} is normalized to be a fraction between 0 and 1.

We can see that first sorting helps with general convergence to real Pareto front, while second improves diversity in the nondominated part of the population. To get the result we just need to extract non-dominated results from the final population.

2.3.2 Spea2

In Strength Pareto Evolutionary Algorithm, unlike previous NSGAii, stores non-dominated indivituals in archive separate from the rest of the population. Therefore after new population is generated, new contents of archive are determined in two passes, first using procedure to ensure convergence and then procedure to ensure diversity.

Let us denote:

P_t to be the population in generation t,$\overline{P_t}$ the coresponding archive of nondominated individuals, N the population size,$\overline{N}$ the archive size and T the maximum number of generations.

The overall algorithm then is as follows: (cite SPEA)

Step 1: Initialization: Generate an initial population P_0 and create the empty archive $\overline{P_0}$, set $t = 0$

Step 2: Fitness assigment: calculate the fittness values of individuals in $P_t \cup \overline{P_t}$

Step 3: Enviromental selection: Copy all nondominated individuals in $P_t \cup \overline{P_t}$ to $\overline{P_{t+1}}$. If size of $\overline{P_{t+1}}$ exceeds $\overline{N}$, then remove exceeding individuals with worst fitness, otherwise, if size of $\overline{P_{t+1}}$ is less than $\overline{N}$, fill $\overline{P_{t+1}}$ with dominated individuals in with the best fitness $P_t \cup \overline{P_t}$

Step 3: Termination: if $t \geq T$ or another stopping criterion is satisfied, then stop and return the result as the nondominated individuals from $\overline{P_{t+1}}$

Step 5: Mating selection: generate the mating pool M

Step 6: Variation: apply crossover and mutation operators to the mating pool and set P_{t+1} to the resulting population. Increment generation counter and go to Step 1.

The final result then contains just the nondominated individuals of the archive.

Fitness assigment

Fitness of an individual is an agregated metric composed of strength of individual and density estimation. Strength of an individual i is the number of individuals from $P_t \cup \overline{P_t}$ it dominates:

$$S(i) = |\{j | j \in P_t \cup \overline{P_t} \wedge i \prec j\}|$$

Raw fitness of i is then agregated from all strengths of individuals that dominate i:

$$R(i) = \sum_{j \in P_t \cup \overline{P_t}, j \prec i} S(j)$$

To distinguish between individuals that do not dominate each other a density metric based on the distance of k-th nearest neighbour of i, denoted δ_i^k. As a common setting, k equal to the square root of the sample size is used (cite spea2). Thus density is defined by:

$$D(i) = \frac{1}{\delta_i^k + 2}$$

Because of the two added in the denominator $0 < D(i) < 1$. Agregate fitness of an individual is then the sum of its raw fitness and density:

$$F(i) = R(i) + D(i)$$

Because the raw fitness has integer values, sorting individuals by the fitness metric yelds similar results to the two pass sorting of NSGAii.

Enviromental selection

Enviromental selecton step depends on the number of nondominated individuals in the union. This is the set of individuals with fitness lower than one:

$$P'_{t+1} = \{i | i \in P_t \cup \overline{P_t} \wedge F(i) < 1\}$$

If $|P'_{t+1}| < \overline{N}$ it is sufficient to set $\overline{P_t + 1}$ to contain the best $\overline{N}$ individuals from $P_t \cup \overline{P_t}$ based on the fitness.

If $|P'_{t+1}| > \overline{N}$ a trimming procedure is used that is based on the density in the P'_{t+1} set, as opposed to $P_t \cup \overline{P_t}$ used for fitness metric.

Iteratively, individual that has minimum distance to another individual is chosen to be removed. If there are several individuals with minimum distance, tie is broken by considering their second smallest distances and so forth.

2.3.3 Ibea

Both NsgaII and Spea2 differ mostly in their aproach to characterize the convergence to Pareto front and diversity. Indicator-Based evolutionary algorithm takes more abstract approach, based on a concept of binary quality indicators. Indicator is a function, that takes two pareto sets of the same domain and outputs some quantification of a difference between the two.

Zitzler and Kunzli proposed two indicators:

ϵ-indicator

Additive ϵ-indicator that quantifies the minimal distance first pareto set has to be moved in each dimension in objective space such that the second pareto set is weakly dominated.Formal definition:

$$I_{\epsilon^+}(A, B) = min_\epsilon\{\epsilon | \forall x_1 \in A, \forall x_2 \in B, \forall m \in M : P_m[x_i] + \epsilon < P_m[x_2]\}$$

If A dominates B, resulting indicator will be negative.

Hypervolume based indicator

Hypervolume-distance indicator, that quantifies how much volume is dominated by the first pareto set but not dominated by the second, with respect to predefined reference point Z. Hypervolume $H(A)$ of a pareto aproximation A is the total volume of n-dimensional space that is "contained" between the individual results and the reference point Z. That is, hypervolume of a set is the total volume of space dominated by the sets individuals.

Hypervolume indicator then compares two pareto aproximations:

$$I_{HD}(A,B) = \begin{cases} H(B) - H(A) & \text{if} \forall x_1 \in A, \forall x_2 \in B, x_2 \prec x_1 \\ H(B \cup A) - H(A) & \text{else} \end{cases}$$

Hypervolume is considered to be the best single value indicator of how good a pareto aproximation is. (c FUNKY HYPERVOLUME THESIIS)

Both of these indicators are dominance preserving.

Definition 7. *A binary quality indicator is denoted as dominance preserving if* $\forall x_1, x_2, x_3 \in Z$*:*

(i) $x_1 \prec x_2 \Rightarrow I(x_1, x_2) < I(x_2, x_1)$*, and*

(ii) $x_1 \prec x_2 \Rightarrow I(x_3, x_1) > I(x_3, x_2)$

Because any single individual can be considered a pareto set of size one, IBEA uses given indicator to quantify fitness of an individual:

$$F(x_1) = \sum x_2 \in P - x_1 a(I(x_1, x_2), k)$$

, where $a(i,k) = -e^{-\frac{i}{k}}$ is function that amplifies fitness of dominating individuals.

Algorithm then goes as follows:

Step 1: Initialization: Generate an initial population P_0, set $t = 0$

Step 2: Fitness assigment: calculate the fittness values of individuals in P_t based on the chosen indicator

Step 3: Enviromental selection: iterate the folowing steps untill $|P_t| < N$:

1. find $x \in P_t$ that has minimal fitness and remove it from P_t
2. recalculate the fitness of the remaining population,

Step 3: Termination: if $t \geq T$ or another stopping criterion is satisfied, then stop and return the result as the nondominated individuals from P_{t+1}

Step 5: Mating selection: generate the temporary mating pool Ma with binary tournament selection on P

Step 6: Variation: apply crossover and mutation operators to the mating pool and set P_{t+1} to the resulting population. Increment generation counter and go to Step 2.

Paradiseo implementation of IBEA uses adaptive scaling of the amplification function parameter, therefore calibration of the k is not as important. On the other hand, chosing appropriate indicator is crucial.

2.3.4 Choosing the algorithm

Design of both of these algorithms give some guarantees on convergence to Pareto front. Both of them are incremental improvements on their predecessor.

Right now, NSGAII is considered to be the standard benchmark for MOEA algorithms. It is the oldest of the three and therefore the most widely used one. Because there are no algorithm specific variables, it has the easiest calibration.

While Spea2 is newer, it doesn't mean it is better in every instance. It has to be noted, that even comparison between these two algorithms with regards to optimizing IDS for WSN (Stehlik 2013) came out inconclusive. On the other hand, several papers (citation) claim they achieved better result with spea2 than NSGAii. Similiary to NSGAii it doesn't have any algorithm specific configuration.

Of the more interesting concepts that IBEA uses is the hypervolume distance indicator, which might provide an interesting alter-

native to heuristics used inmore popular NSGA and SPEA. Unfortunately experimental results in (citacion) show, that I_{HD} based IBEA is very sensitive to miscalibration of reference point Z. If provided with good reference point, it consistently outperformed the other tho algorithms, but optimal value of Z varied greatly based on the problem set. We will tallk more about hyper-volume as a metric of pareto set convergence in chapter on calibration.

3 Our Framework

(b Stetsko 2011) showed, that using simple evolutionary algorithms is a viable and efficient way of optimizing Wireless Sensor networks. Unfortunately, prevoius experimental setup was too tightly coupled with optimization framework, rendering the code of previous experiments hard to reuse. Therefore we have created a micro-framework based on python and ParadisEO.

Our micro-framework simplifies its usage by allowing to specify evaluation, mutation and crossover functions in python. Emphasis is on multiobjective algorithms, ability to evaluate population in parallel and having good facilities for experiment analysis.

3.1 Paradiseo MOEO

ParadisEO is a highly configurable framework for metaheuristics written in C++.

Out of the seven algorithms provided in Paradiseo, we include three of them: NSGAII, SPEA2 and IBEA. We do not include the predecessors, NSGA and SPEA, because in general, they have worse performance. SEEA, or simple elitist evolutionary algorithm is best suited for problems with inexpensive evaluation functions, where the most time consuming process is the selection, but evaluating WSN configuration is seldom cheap. Should for any reason be an algorithm not included in our micro-framework be better suited than the ones that are, it is easy to add one. Main modifications required are to enable parallel evaluation for the algorithm.

Even if the algorithm we'd wanted to use wasn't present in Paradiseo framework, due to its whitebox nature and modular design and large library of integrated metrics it could be easily added.

3.1.1 Object-oriented design of MOEA

Every multiobjective evolutionary algorithm in Paradiseo extends the moeoEA. The implementation of such algorith is best explained on a moeoNSGAII[CITE], probably the most popular multi-objective algorithm. Because moeoEA implements the unary functor interface,

to define new algorithm it is sufficient to define the the operator().

As you can see, the algorithm itself is a template defined by implementation of the fitnessAssigment,diversityAssignment,breed,popEval, replace and continuator functors and eoPop population object. Therfore most of the implementation iself is done in definig various classes that implement. We can showcase how this composing works on an example of a continuator:

If we use constructor that ends evaluation after certain number of generation, it first initializes defaultGenContinuator with number of generations, and then passes it as a continuator to use.

Population object

Population in Paradiseo is in its core an ordered vector of individuals. Because our individuals are represented by vectors of real numbers, we will describe the moeoRealVector implementation. Each individual has its value, and contains variables for fitness and diversity metrics. It should be noted, that moeoRealVector doesn't contain any bounds on specific parameters of individals configuration. In paradiseo, we define boundaries externaly, with specification in initialization function and crossover and mutation functions.

Fitness and diversity assigment

Class moeoDominanceDepthFitnessAssignment is used to sort the population into nondominated ranks, by updating the fitness variable of each individual. Class moeoFrontByFrontCrowdingDiversityAssignment is similarily used to set the diversity variable. The replace operator uses these to evaluate the children of the old population and creating the new generation.

Creation of new generation

Breed operator is a standard eoBreed class that gets supplied with transform-object that agregates mutation and cross-over functions. Similiarily, the popEval function just evaluates all the new individuals in populations and assignes them objective values. The specifics

of the algorithm are hiden in the replace operator of moeoElitistReplacement class:

1. recalculate fitness and diversity
2. sort the population with standard moeoFitnessThenDiversityComparator
3. remove exceeding population

As you can see, this implementation slightly differs from the sketch of algorithm provided in previous chapter, namely that the NSGAII in paper never exceeded the population size.

Stopping criteria

Continuator operator provides the basic stopping criteria. The default implementation is based on limiting maximum number of generations[1], but in theory, continuator can use population fitness or another metric to decide whether to proceed with another step. This feature is often utilized in evolutionary implementation of decision problem algorithms, because it can help evaluate whether population contains enough information to carry out the decision. In multicriteria optimization problems we usualy don't have a definition of "good enough" solution, but if we had, we could use it to further limit the number of expensive evaluations.

We chose python for these reasons:

1. it is an efficient glue language. Most often you will want to optimize already written application with minimal changes. Python is well suited for running external applications, specifying their inputs and parsing their outputs.
2. it has good facilities for statistical analysis. That allows us to include analysis of every optimization run into the executable itself. Having automatically generated experimental log has proven invaluable especialy in callibration of algorithms for our spepcific problem.

1. As hinted by the class-name *defaultGenContinuator*

3. it is easily integrated with C++. We strive to make the integration with ParadisEO simple and extensible.

Our microframework consists of a single executable and a configuration file in python. Configuration itself consists of definig selected entry points, in particular the number and bounds of input parameters, the number of objectves to optimize, definitions of crossover and mutation functions and either the definition of evaluation function, or functions for sheduling evaluations in parallel on boinc. We provide reporting function entry-point as well, to automatically colect and analyze results of the algorithm run.

3.2 Entry points

3.2.1 Input parameters

For inputs you specify their bounds, which are then used to initialize the population and to supply constraints for mutation and crossover parameters

3.2.2 Objectives

Because we focus on multiobjective evolution, you need to specify number of objectives. To simplify configuration,minimization of all the objectives is hardcoded.

[example]

It is advisable to have three objectives at maximum, because number of solutions on the Pareto optimal dramaticaly increase with new objectives. At minimum, $(n+1)$ objective problem will contain all the solutions of an n-objective problem. [CITATION? 309 talbi]

Then there is the problem of visualizing and evaluating more than 3-dimensional data.

3.2.3 Evaluation function

To simplify configuration, inputs and outputs of evaluation function are allways an n-touple of floating point numbers. We believe this is general enough, and that most WSN optimisation problems can be

fitted to this constraint. Only problem might pose translation of combinatorial problems to continuous ones, fortunately so far we have been quite successfull with using just a simple rounding techniques. We wanted to avoid usage of binary strings as input vectors.

Many evolutionary algorithms support them and if we were using them we wouldn't need to concern ourselves with implementation details of mutation and crossover functions. On the other hand, with generalized mutation functions on binary strings, there are no guarantees that mutated individual will even be valid. CITATION In our opinion, vectors of reals are esier to reason about, and their convergence may provide valuable insight to structure of the problem.

3.2.4 Boinc Assisted Evolution

Because evolution is population-based heuristic, it is well suited for parallel evaluation of its individuals. ParadisEO already has two methods included, either by use of MPI protocol [CITATION], or by using SMP on multi-core machine. Unfortunately, SMP module of ParadisEO is not yet stable on all platforms, and MPI has specific demands on infrastracture and is usually hard to retro-fit on already written program.

Boinc on the other hand can simply utilize machines already preasent to form a simple computational grid. With its simple archhitecture consisting of Management server and several worker nodes it can be deployed in most settings. Its simple architecture prohibits cooperation between worker nodes, but that doesn't concern us while evaluating individuals.

We have decided to give the evolution algorithms ability to use a simple scheduler for creating Boinc work units, plugable from python.

In our micro-framework it consists of three functions, schedule_work_unit, wait_for_completion and gather_result.

Even though the evaluations themselves will be done in parallel, we wanted to avoid any threaded code in our framework itself. Because we have written it with research in mind, we understand that accessing auxiliary data is important to evaluate experiments. In threaded environment, this could lead to deadlocks.

3.2.5 Reporting

Reporting function takes two arguments, both strings. First string is the char * args string passed to the executable. Second string is the console printout or the Paradiseo result-archive.

This way, we could integrate the creation of experimental log directly into our executable, computing different . This helps especialy with calibration of evolution algorithm parameters. We provide examples of different metrics and indicators that help with evaluation of result in next section.

Integrating python with C++ by passing strings is not particulary elegant. We have chosen this particular implementation, because it is easiest to extend and debug.

3.3 Calibration

We can think of calibration on an evolutionary algorithm as a meta-optimization problem with two objectives,

1. maximizing the fittness of result
2. minimizing the number of evaluations

With single-objective algorithms, optimizing for these two objectives is easy, because both of them are usualy represented by numbers. With multi-objective problems, resuls are Pareto sets, which are often not directly comparable.

Quality of a multiobjective result can be judged based on several criteria:

1. number of results in Pareto set.
2. convergence tp true Pareto front
3. regularity in distance beween results

3.3.1 Exhaustive search

Because the main reason we use multi-objective algorithms is to avoid the need to do the expensive computation of true pareto front, we

could calibrate it on a easier variant of the problem. Underlying assumption is, that the particular variant of the problem will have similar solution space. We will show such an exampe in following chapter.

Computing an exhaustive search for a subset of our problem can help in several ways:

- we can check our assumptions about the shape and homogenity of solution space
- we can use the Pareto front as a reference point
- for any approximation, we can easily compute how many individuals it dominates and how many individuals dominate is

This is probably the most the most expensive, but on the other hand the most precise way to estimate, how will an evolutionary algorithm behave in a particluar problem domain.

3.3.2 Iteratively

For iterative improvements we have used different convergence critera. To measure convergence of Pareto aproximations to the Pareto optimal set, we can either specify a global reference point, with respect to which all approximations are measured, or we specify a comparison function that can provide ordering.

Measuring convergence

Hypervolume indicator One of the more popular is maximizing the hypervolume of the objective space dominated by the resulting approximation. Hypervolume indicator is based on idea of measuring the volume that a given individual dominates based on some reference point in the solution-space. This works with assumption that the solution-space is homogenous, compact, and that we can specify a reasonable reference point. A good reference point is usualy the worst individual [CITATION NEEDED] and because because we usualy don't know how will this individual look like, by guessing this reference point we are expresing certain biases (or expectations) on the shape of the solution space as well.

Distance indicator Distance indicator $I_{\epsilon+} : Z \times Z \rightarrow \mathbb{R}$ is one of possible choices for indicator that IBEA algorithm uses for comparing two pareto-approximations. $I_{\epsilon+}$ gives the minimum distance, by which Pareto set approximaiton needs to or can be translated in each dimension in objective space, such that the other approximation is weakly dominated.

3.3.3 Measuring diversity

Diversity metrics help us avoid pareto approximations, that have too similiar solutions. We used two metrics, one taken from NSGAii, the other from Spea2.

Crowding distance metric This is the metric that NSGAii uses to ensure diversity of next generation.

Spacing metric This is the metric that SPEA2 uses to ensure diversity of the result-archive.

3.3.4 Population size and Number of generations

These two parameters set the boundaries on minimal and maximal number of evaluations. Lets mark the size of population N and number of generations G Because the whole initial population needs to be evaluated, number of evaluations will at least the size of populations. Because at each new generation, at most N new individuals are generated, therefore in one optimization run, at most $N \times G$ evaluations.

3.3.5 Mutation and Crossover

Probability of mutation, or crossover directly influence number of generated individuals in each generations.

4 Experiments

To test our framework we have decided to optimize a simple intrusion detection system on a model wireless sensor network.

4.1 Wireless sensor network

Implementation of WSN was reused from previous research (Stetsko 2012), with just a slight modifications to decouple it from previous optimisation framework. Nodes are simulated with MiXiM framework, with test application on each of them, that sends a packet containing arbitrary information through the network to a base station. Network has a static routing tree based on TODO algorithm.

We have marked some of the nodes as "droppers", so that instead of forwarding all of the packets through the network, they drop certain percentage. In our case, dropping ratio was set to 0.5.

4.2 Intrusion Detetection System

Our nodes implement a simple IDS capable of detecting nodes that seem to be intentionaly dropping packets. If we look at the wireless network stack of our node, IDS is part of the medium access controll sublayer. There it can eavesdrop on neighbouring nodes and check whether they behave accordingly.

For each neighbour it updates

- number of packets recieved
- number of packets forwarded.

We wanted to use IDS that is simple, but highly configurable. We have these four parameters to optimize:

1. number of monitored nodes
2. size of buffer for eavesdropped packets
3. treshold for minimal number of recieved packets for a node to be considered malicious

4. treshold for ratio between forwarded and recieved packets for a node to be considered to be malicious

At the end of simulation, based on a preset tresholds and ratio between forwarded and reieved packed it decides whether node is malicious or benign. More specificialy, the set of malicious nodes is:

$$M = \{n \in N | r_n \geq r_{min} \wedge \frac{f_n}{r_n} \geq d_{max}\}$$

And the set of benign nodes is:

$$B = \{n \in N | r_n < r_{min} \vee \frac{f_n}{r_n} < d_{max}\}$$

You can see, that both sets are disjoint, and their union is the set of neighbours N

4.3 Setup

We want to use our framework to optimize IDS in this boundaries:

Because this experiment is aimed primarily on showcasing the functionality of our framework, we have first calculated a two sample exhaustive searches to gauge the problem space and run calibration on it, that are not as resource intensive.

After somparison of problem spaces, we have chosen the best settings and algorithm. We then apply it to several, form computational resources standpoint, expensive problems. With the help of Boinc-assisted evolution we could assess whether MOEA evolution can provide reasonable and consistent optimisation of IDS on WSN.

All the algorithms will share mutation and cross-over functions:

Mutation function

Our mutation function had two parameters,probability that individual will be mutated $pMut$ and relative δ neighbourhood around the old value, that the new value will come from:

TODO CODE,IMAGE

Crosover function

Our cross-over function, with parameters $pCross$ (probability that two individuals will be crossed over) and $crossProb$ (probability that params of the two individuials will be swapped):

Calibration

For algorithms NSGAii and Spea2, calibration was done on fixed number of generations and population size, with optimizing for values $pMut$, δ, and $pCross$.

For IBEA we have chosen the I_{HD} indicator, that we use as a comparison metric for other two algorithms as well. Because IBEA with I_{HD} requires calibration of reference point Z, we haven't used crossover operator with IBEA at all ($pCross = 0$).

4.4 Results