System-of-Systems Operational Analysis and Optimization
A Case-Study of Quadcopters fleet in Real-Time Delivery Mission

Chana Ross∗ and Tomer Rokita†
Rafael – Advanced Defense Systems Ltd, Haifa 3102102 Israel

This paper describes an operational analysis and optimization of a quad-copters fleet, designed for real-time package delivery. The fleet and its mission serve as a benchmark problem, nicknamed Rafazon, of a complex system-of-systems (SoS). First, a computational model was established using agent-based discrete-event-simulation (AB-DES). Next, the simulation was integrated into a multi-objective-optimization framework, in order to efficiently find the optimal fleet composition, which maximizes system efficiency, while minimizing overall cost. Finally, the optimization results are presented as a set of pareto optimal system designs, which reveals the SoS design trade-offs.

I. Introduction

Many systems today include a collection of task-oriented or dedicated sub-systems that pool their resources and capabilities together to create a new, more complex system which offers more functionality and performance than simply the sum of the constituent systems. These complex systems are usually referred to as system-of-systems (SoS). The definition of the optimal design of such SoS poses a challenge for classical system engineering methods.

An example of an emerging market that utilizes SoS is the commercial use of Unmanned Aerial System (UAS), which has the potential to reshape the delivery market and other markets attempting to use autonomous vehicles for transportation. UAS have been widely integrated into the defense market [1] and are now being shifted and used for commercial delivery companies (such as UPS, Amazon, local stores, restaurants etc. [2–4]). In this paper, a new benchmark problem, nicknamed Rafazon, is defined as a set of requirements for a "real-time" package delivery system using a quad-copters fleet. The benchmark problem is used to demonstrate the developed methodology for simulating and optimizing the SoS working point.

Consider the problem faced by a UAS dispatcher who must plan for a set of UAVs to service a set of requests and assignments, which are stochastic in time, and arrive in a dynamic fashion (the system does not know of future requests when starting). To service a request, one of the UAVs must fly to the location of the request and perform a known action (unload a package, observe the wanted point etc.). Once the assignment is finished, the UAV can continue to the next assignment or return to the base to recharge (the dispatcher must give the UAV the following command). The dispatcher wants to minimize the cost of using the UAVs while achieving the maximum system efficiency. Even if the UAVs aerial models are deterministic and the routes chosen per assignment are known, the stochastic arrival of requests can make for a very challenging on-line planning problem. In addition the UAVs capabilities have a major effect on the efficiency of the system and therefore need to be accounted for. Due to the complexity of the systems and the huge amount of combinatorial combinations that define the SoS, there is a need for a new method to understand and analyze the system and the different disciplines within it. The proposed method relays on an efficient simulation, that enables the search of several optimal working points for the system.

The presented approach for simulating SoS is based on Agent-Based Discrete-Event-Simulation (AB-DES), which was implemented based on commercial engine called AnyLogic [5]. Once the simulation is established, it is integrated into an optimization framework to drive the design variables for minimizing or maximizing design goals (for example: minimize overall cost, maximize system efficiency etc.). In the current work, a multi-objective optimizer provided by modeFRONTIER software (by Ésteco [6]) was used.

∗Multi-System Analysis & Global optimization (MUSTANG) group, chanaby@gmail.co.il
†Head of Multi-System Analysis & Global optimization (MUSTANG) group, Associate Member AIAA, tomerro@rafael.co.il
II. Benchmark Problem

The SoS benchmark problem, nicknamed Rafazon, is described in the following sections.

A. Background Story

Rafazon is a leading internet-shopping retail company. Rafazon’s Prime members are enjoying a two-day-delivery service for a wide variety of products; this service has become the company trademark. The company currently considers the deployment of an innovative and market-breaking project, in which they will introduce a new exclusive Prime2min membership. Prime2min members will enjoy a two-minutes delivery (allowing a minute delay if needed) of three main products (product A, B & C). This service will utilize a fleet of quad-copters holding the products in the air, in proximity to the Prime2min members. At first, this service will be launched and tested only in down-town Tel-Aviv city, and might expand to other major cities in the future. The area of deployment is about $3_{km} \times 3_{km} = 9_{km^2}$. Based on their market review, Rafazon expects up to 4000 orders per day in the entire deployment site, distributed equally spatially.

Three types of quad-copters are available ($X, Y, Z$, each with different capabilities and prices, as described in Table 1). There are three main products ($A, B, C$, each with different size, weight and number of requests per day with known distribution function). Product A weighs $0.25_{kg}$, product B weighs $0.5_{kg}$, and product C weighs $0.75_{kg}$.

<table>
<thead>
<tr>
<th>Table 1 Quad-copters specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>Max. payload [kg]</td>
</tr>
<tr>
<td>Hover time full [min.]</td>
</tr>
<tr>
<td>Hover time empty [min.]</td>
</tr>
<tr>
<td>Price [k$]</td>
</tr>
</tbody>
</table>

B. Research Questions

Rafazon is interested in finding an optimal solution to their new service, which would enable 2-min-delivery 24/7, at a minimal cost. Specifically, the company asks the following questions:

- What type of copters should the company use? (expensive copters with high endurance and large payload weight or cheap copters capable of flying fast with small payload?)
- How many packages should each copter carry?
- How many copters are required in order to have maximum delivery efficiency?
- Where should the copters base be located? (Taking into account that the base is large, and as it is closer to the delivery site the real estate price is higher)
- What should the copters do when waiting between deliveries? (go to central location/ fly to random location/ hover to waste minimum battery percentage etc.)
- When a package is requested, which copter should deliver the package? (closest copter/ copter carrying the most packages/ copter with minimum battery left)

In addition to the general questions asked about the system and its features, there are three main objectives that need to be addressed:

- Minimum total system cost
- Minimum number of quad-copters
- Maximum product availability

III. Theoretical Background

Many previous studies solve these type of problems using theoretical equations with many assumptions and reductions to the real problem [7][9]. We on the other hand assume that complex optimization problems which take into consideration all the constraints and objectives cannot be solved analytically and therefore a simulation is built in order to represent the objective function and constraints. The process for solving the simulation-based optimization problem is as follows:

- Choose design variables
• Run simulation (in order to capture the stochastic behaviour of the system and agents)
• Based on output parameters change the design variables and re-run simulation

This process is done using an optimization algorithm until the Pareto front is found and sufficient results have been achieved. It should be noted that the simulation must run repeatedly hundreds if not thousands of times and therefore needs to be computationally efficient.

The current problem is divided into three sub-problems:
• **Planning problem** - assuming a given environment and agent capabilities, how should the manager allocate assignments throughout the fleet?
• **Simulation** - what type of simulation should be used and what should the simulation architecture be in order to describe an instance of agents and environment acting and interacting with one another throughout a given time span.
• **Optimization problem** - assuming a given planner and simulation, how many agents are needed, what should their capabilities be and what should the environment be (base location, flight height or any other parameter that can be determined in the simulation).

In the following sections we will define the theoretical background for the three sub-problems.

**A. Planning Problem**

While most research in the field has concentrated on off line scenarios (known deterministic scenario and vehicle performance), the current problem can be defined as an "Online Continual Planning Problem" (OCPP [10]), where there is a number of agents (quad copters in the current case) which can perform different tasks (vary with each scenario), while requests arrive at unknown times during the simulation. During the simulation the agents manager must decide which request to answer at each given time. We define the OCPP as a Markov Decision Process (MDP [11]) with both world states and goal sets. The OCPP consists of a 5-tuple P=<W,G,A,T,C>, where W is the set of world states, G is the set of goals (requests) created within time, A is the set of actions that can be chosen at each time step in the simulation (combination of all quad copter actions). A state S is the combination of world state (W) and current goals opened (G). The function T is the transition function of request arrival distribution. Finally C is the cost function which defines the cost of performing action A at a given state.

For each simulation the total cost is:

$$C_T = \sum_{i=1}^{T} (C(s_i, a_i))$$

our goal is to minimize the total cost under different constraints. In addition we are interested in running different policies and comparing the optimal systems found for each policy and the behaviour of the system under these policies. In the next chapters it will be shown how each policy effects the performance of the system and the requests answered throughout time.

**B. Simulation**

When looking at complex systems like the one described here, the policy or planner is only one part of the whole equation that needs to be taken into account when trying to define an optimal SoS. In order to fully understand all the different aspects of the problem we need an efficient simulation.

Problems similar to the one at-hand can be simulated using three main methods:
• System dynamics simulation
• Discrete-Event-Simulation (DES)
• Agent-Based-Modelling (ABM)

Each simulation can be used for different purposes, and many times the same problem can be simulated in all methods while having a different purpose for each simulation.

1. **System Dynamic Simulation**

When interested in understanding a specific subsystem or discipline within the overall problem a System Dynamic Simulation is typically used. This type of simulation is time based using differential equations to describe the problem (either very dense discrete time-line or a continuous one). The simulation is usually accurate in regards to the subsystem described and general regarding the rest of the system. In addition the run time is very slow since each time step needs
to be calculated. An example for this simulation can be the control system of a quad copter during a specific mission such as take off and package delivery.

2. Discrete-Event-Simulation

Unlike System Dynamic Simulations, Discrete Event Simulation (DES) are event based and usually used to describe processes or complex system pipelines [12]. These simulations are used in many engineering problems in order to describe an MDP (Markov dynamic process [?]) which is more complicated than the basic dynamic processes. DES simulations are generally used similar to factory packaging, healthcare, supply chains and more. In order to describe a problem using DES, one needs to assume a repetitive process that can be easily described using basic logic since the simulation is driven by certain events occurring throughout time. In DES, each discrete state is updated based on asynchronous events that occur while the state of each component is changed dynamically throughout the simulation.

DES focuses on the entire system process, while usually each component is passive within the overall pre-specified process. A classic example for DES is a packaging process, in which each package has a loading start time, end time and size. These parameters define how long the full process will be busy with each package and will define the overall time-line of the simulation, since the dynamic behaviour of the simulation is "event driven". DES time-line can be defined based on asynchronous events or run at a specific frequency while making sure the frequency is smaller than the event duration.

3. Agent-Based-Modelling

When considering complex systems (each agent can perform multiple tasks, all agents work together on a joint mission, goals are effected by agents decisions and many more), DES can not be used to describe the system, and therefore a more advanced approach is needed.

Agent-based-models are the solution since they are a more advanced version of DES where the agent is the main driver of the simulation and not the events [13]. The main focus of these simulations is the entities themselves and the relations between them while each entity makes autonomous decisions (or decisions based on a controller agent that gives out commands to its fleet). Similar to DES, the time is discrete and defined by agents behaviour. The main difference between ABM and DES is the dynamic capability of the agents within the system. In DES the agents do not make decisions and continue in their states with known actions pre-defined; in ABM the agents choose their actions based on their policy and the actions can be stochastic or unknown to the full model.

Overall, for the current purpose ABM is the most useful while some of our subsystems or processes can be defined using DES or system dynamics (for example, the system dynamics approach is used for describing the energy of a quad copter, since the quad copter uses up energy in a continuous matter). Such an hybrid approach is available in the Anylogic software [5].

C. Optimization Problem

Unlike most optimization problems, in the current case the objective function can not be defined as a mathematical equation, but it is the output of the simulation. Once the simulation is created, we choose a number of design variables that define the problem and let an optimization algorithm find the set of optimal results for these design variables. The formal definition of the optimization problem is as follows:

$$\max_{x_1, x_2, \ldots, x_n} \quad F_i(x_1, x_2, \ldots, x_n), \quad i = 1, \ldots, k.$$ 

subject to

$$c_i(x) \leq b_i, \quad i = 1, \ldots, m.$$ 

where $x_1, x_2, \ldots, x_n$ are the design variables, $F_i$ are objective functions, and $c_i(x)$ are constraint functions.

The optimization problem at hand is characterised by:

- The design variables can be integer (number of agents for example) or continuous (quad copter capabilities)
- Multi objective optimization
- Stochastic objectives and constrains (for each set of design variables chosen by the optimization algorithm, the objectives and constrains are not deterministic - the output is usually represented by a mean and standard deviation)
- Large combinatorial space of the design variables

These characteristics of the optimization are not standard and therefore methods like gradient descent [14] or other known optimization algorithm can not be used. In order to solve the optimization problem the optimization framework of modeFRONTIER [6] is utilized. This framework allows integration of different computational software’s (any
commercial or in-house code) into a common design environment, thus allowing the automatic run of a series of computations proposed by a selected optimization algorithm, until the specified objectives are achieved.

In this modular environment, each component of the optimization process (input variables, input files, scripts, software, output files, output variables, objectives and constraints) can be defined as a node and connected to the other components in order to create the optimization logical order. Once the logic is defined and the optimization flow is created, different optimization algorithms can be used to find the Pareto and optimal results. Once the optimization algorithm is run and results are shown, Statistical and Multi-Variate Analysis tools are applied to extract important information about the system.

IV. Computational Experiments

The solution procedure was developed using AnyLogic for the simulation and ModeFrontier for the optimization. The following chapters include an in-depth description of the different policies examined, the simulation structure in AnyLogic, and the Optimization framework.

A. Policy

Per a specific configuration and simulation there are many decisions the manager needs to take. The simulation is built in a way that each copter does not choose its own assignments but rather the manager has a matching algorithm between all the assignments and resources. This algorithm is performed whenever a new request is created or a resource becomes available. In addition, if a copter needs to return to base, the manager searches for copter to replace it (if one exists) or adds the copter to a list needed to be replaced. The policy algorithm used in the simulation can be part of the optimization parameters and could be analyzed at any stage of the research. Two main policies are chosen for comparison:

- **Hub policy**: This policy assumes each copter flies to the hub when reaching the area and once finishing an assignment. This is beneficial when copters perform better while hovering, since most of the time the copters are idle, they will be in hover state. In addition this policy assumes requests are created in a known area close to the hub location.
- **Random policy**: This policy assumes copters fly randomly as long as they are in the area and not assigned a specific assignment. This policy is useful when requests are distributed randomly in space since it allows copter to explore random areas and be close to events in all areas when needed.

In the results section the results of a simulation with both policies will be compared.

B. Simulation

Anylogic is java based and includes agents that are java classes. Each agent can perform multiple state charts and can have a visual representation in the environment if wanted.

In order to have re-usability of the simulation for different problems and levels of model-fidelity, we created a layered simulation where each agent includes a number of capabilities (Move, Deliver, Energy etc.). Each capability is an agent as well, with state charts that define its logic. The agents communicate with one another using messages where each message includes a text representing the command given by one agent to another, and any other parameters required to accomplish the command.

The simulation is built of three types of agents:

- **Static agents**: these agents are defined at the beginning of the simulation and create the environment in which the dynamic agents exist in.
- **Logical agents**: these agents define the logic within the simulation and manage all the choices of the dynamic agents within the simulation.
- **Dynamic agents**: these agents are the real within the simulation. They react to the static agents and other dynamic agents based on the logic given to them by the logical agents.

The following agents exist in Rafazon simulation:

- **Static**:
  - Zone: this agent defines the dimensions and location of the zone where the packages are delivered.
  - Base: this agent defines the location of the base compared to the zone.
- **Logical**: 
- Main: this agent includes all the agents within the model. In addition all statistical calculations are performed in this agent.
- Manager: this agent includes all the logical functions which exist in the simulation. Every change in the environment or in a dynamic agent starts a process in the manager which decides how each dynamic agent will react to the change. Once the manager chooses which agent should react to the following change it starts a process in the chosen agents.

**Dynamic:**
- Copter: the copter has different capabilities (flight cycle, move, deliver, energy). Each capability is defined as its own agent with separate state charts for each logic within the capability. The copter reacts based on its inner logic (battery consumption for example) and on messages it receives from the manager (command to deliver package).
- Order: this agent starts when the manager creates it and includes the basic capabilities of the order. In our simulation the order is passive and has basic states defining its aliveness.

1. **Quad Copter**
   The quad copter is a java class which includes other java classes (Agents) in it, each representing a different capability. The capabilities simulated in the current case are:
   - Move: This controls the copters moving capabilities assuming a given location. In Figure 1, one can see the logic of the move capability. Once a message is received from the manager with the next location the move agent updates the copters velocity and destination and sends the agent a message to move to the wanted location. The state chart returns to idle when the location is reached, or if the command is terminated by the manager or other controllers of the copter (energy ended for example).
   - Deliver: When copter is assigned a request to deliver this capability moves the copter to the requests location and then unloads the package. In Figure 2, one can see the logic of delivery once a controller sends the copter a command to deliver a package. The copter first fly to the delivery location then unloads the package, and finally returns to idle and waits for another command. It can be seen that at every stage of delivery the process can be terminated by an external command and the package would not be delivered.
   - Energy: This controls the copters energy state at every given time. updates when copter releases a package, flies or reaches the base for recharging. In Figure 3, one can see the energy logic, assuming the copter is fully charged when leaving the base and the energy is lowered based on the current discharge rate (the discharge rate is a function of the number of packages the copter is carrying, the velocity and the type of copter). Once the copter reaches its energy limit it sends a message to the flight cycle logic to return to base. In addition the copter sends the manager a message with the same information, this is for maintaining a steady state number of copters in the area at every given moment possible.
   - Flight cycle: This controls the basic flight cycle of a copter from base -> area -> base assuming the copter needs to recharge when battery reaches a limit. Figure 4 shows the overall logic of the copter once chosen to go to the area and be active in the delivery process. Once the copter receives a command to fly to the area it performs different actions in the zone until sent back to the base to recharge by the energy logic.

All these agents together create the full logic of the copter. They all communicate with each other and each agent is responsible for a different aspect of the copter.

2. **Manager**
   A manager is needed for controlling all system level decisions. The manager includes the following decisions:
   - Assign quad copters to opened requests (based on a given policy)
   - Find quad copters to replace quad copters going to base to recharge
   - Send quad copters new assignment when in area
   - update requests they are finished once the quad copter releases their package
   - Generate new requests based on a known distribution function

The manager is implemented using state charts, where each state chart represents a different assignment the manager is in charge of.
Fig. 1  Move agent state chart

Fig. 2  Delivery agent state chart
Fig. 3 Energy agent state chart

Fig. 4 Flight Cycle agent state chart
3. Requests

The requests in the simulation are static and do not have any internal logic other than being created and answered/answered late or canceled. The requests status is updated to answered based on updates from the manager. If no update has come from the manager the request is canceled based on a known time given by the user at simulation start time. In Figure 5 the full logic of a request is shown.

C. Optimization

The optimization framework is created within ModeFrontier, and then run with different algorithms and optimization objectives. The NSGA-II [15] (non dominated sorting genetic algorithm II) algorithm, which is an evolutionary algorithm for multi-objective problems, was chosen. Evolutionary algorithms were developed because the classical direct and gradient-based techniques have the following problems when leading with non-linearities and complex interactions:

- The convergence to an optimal solution depends on the chosen initial solution
- Most algorithms tend to get stuck to in a suboptimal solution.

In contrast, NSGA-II has the following three features:

- It uses an elitist principle, i.e., the elites of a population are given the opportunity to be carried to the next generation.
- It uses an explicit diversity preserving mechanism (Crowding distance)
- It emphasizes the non-dominated solutions.

These features are useful in the current case, where exploration is needed but also the simulation is rather costly and therefore we cannot run infinite amounts of points in order to fill in the pareto.
V. Results and discussion

The results can be viewed on three levels:
- Single copter performance
- System performance
- Optimization results

Each level is examined separately in order to better understand the system as a whole.

A. Single Copter Performance

Each copter velocity and discharge rate changes depending on the state the copter is in (charging when at base and discharging while flying or hovering). In Figure 6, one can see the discharge rate and total energy left of a specific copter versus time. We can see that every time the copter reaches the request it needs to deliver it goes into hover mode and the discharge rate is changed. In addition, the energy level changes linearly when flying/hovering and is regained when copter reaches the base to recharge. These graphs represent the typical behaviour of the copters capabilities in the simulation while each copter type differs by the discharge rate and charge rate, given to it by the user. In addition each simulation and copter has a slightly different pattern depending on the capabilities given by the user and the assignments it preforms during the simulation.

At the beginning of the simulation each copter is loaded with a number of packages to carry from the base when flying to the area. This number can change and has a maximum capacity based on the specific copter capability. The number of packages effects the copters performance and it can be seen in Figure 6 how the discharge rate changes once the copter releases a package. In addition the copter has a policy of idle time. We will see in the following section how different policies effect the total system performance.

B. System Performance

In order to maintain continuous supply, the manager tries to keep fixed number of working copters in the area (working copters are ones that have more than one package on them and enough energy to reach requests if the manager commands on them to deliver). When the simulation starts, k copters are sent to the area at staggering times, such that when the last copter is sent the first goes back to base to recharge. This allows for a continuous cycle of copters between the area and base.

In Figure 7, the number of copters in area and in base during the simulation are shown. The x-axis is the time from the start of the simulation and the y-axis is the number of copter in base (top figure) and in the area (bottom figure).

Once the simulation reaches a steady state, the average number of copters in the area is equal to the number defined...
by the user. One can see fluctuation in the graph because the system is sub-optimal and during peaks of request arrivals, there are not enough copters for the full assignment, and therefore more copters need to go back to base and recharge or get more packages.

Fig. 7 Number of copters in area and base vs. time

In Figure 8, one can see the cumulative number of requests created over time and supplied. The figure indicates the total number of requests, number of requests supplied on-time, late and cancelled. This figure is the basic figure which describes how efficient a specific system is. The more requests supplied, the closer the combined delivery line will be to the total number of requests. In addition, the efficiency of the system can be extracted from the last time as:

\[ \text{Eff} = \frac{N_{\text{late}} + N_{\text{on-time}}}{N_{\text{Total}}} \]

Fig. 8 Number of requests created, delivered, delivered late and cancelled vs. time
Once the simulation is created and validated, different features and policies of a specific scenario can be examined. One of the interesting points in the current research is the effect of policies on the system as a whole. For example: "What should copters do while waiting in the area for their next assignment?". Next Hub policy is compared with Random policy, as described in Section V.A.

In Figure 9 one can see a comparison in a specific simulation between both policies. Each dot in the figure represents the location of a request in the simulation, while the color indicates the requests status (red - canceled, purple - delivered late and green - delivered on time). We can see that the figure with the hub policy has a clear preference for requests near the hub whereas the random policy responds to all requests equally. This is one example of the strengths an AB-DES simulation has. More research questions can be answered with the simulation without preforming an optimization.

C. Optimization Results

Once the simulation is completed and can be run per a specific configuration, the optimization loop is created. The input variables are varying depending on the optimization objectives and the working assumptions. The Pareto front for each optimization can be examined, while analyzing the composition of specific configurations along the Pareto. In addition, the system-level trade offs between the objectives are revealed.

This paper focuses on two optimization problems:
- **Constant base location**: What is the maximum system efficiency possible with the minimum amount of copters, for a given base location? This problem has 9 variables (number of copters in area of each type, number of copters in base for backup of each type, and number of packages on each copter). There are 2 objectives (amount of copters and efficiency).
- **Fulloptimization**: What is the maximum system efficiency with the minimum amount of copters and lowest price? This problem has the same 9 variables as the previous problem with an added variable for the base location (the closer the base is to the area the more efficient the copters are but the more expensive the real estate is), and 3 objectives (amount of copters, price and efficiency).

1. **Constant Base Location**

In Figure 10 one can see all results from the optimization run. Each dot in the figure represents another configuration chosen by the optimization loop and its simulation result. The x-axis is the total price, the y-axis is the system efficiency and the color represents the number of copters (in base and area of all types). Most runs in the figure are close to the Pareto front and not wasted in areas that are sub-optimal. In order to understand the optimization results the Pareto front is examined in Figure 11. This figure represents the optimal solutions the algorithm has reached to. Unlike regular optimizations there are many optimal points because the current problem has more than one objective. Each point on the Pareto figure shows a solution that is best for one of the objectives and limited by the other ones. (for example there is no configuration that reaches more than 80% efficiency with 30 copters or less). In addition, a clear correlation
between the copters price, number of copters and efficiency can be noticed from the figure (as the price rises, there are more copters and the efficiency goes up). It can be seen that in order to achieve 100% efficiency, at least 60 copters are required. Comparatively, if the required efficiency is reduced to 80%, then the amount of quad-copters could be reduced by less than half.

Fig. 10 Optimization Results

Fig. 11 Pareto Result - Constant Base

2. Full optimization

In Figure 12 the results of the full optimization can be seen. Unlike Figure 10, the color in this figure represents the base distance from the area. One can see that with fixed number of copters, as the base is closer to the area, it increases system efficiency. This makes sense since the copters are more efficient and can distribute more copters on each cycle when the base is closer to the area (less time is wasted on the transportation from the logistic base to the area). This optimization is more complex since each base location chosen creates a similar problem to the previous one where the base location was constant. In Figure 13 the Pareto front for this problem can be seen, unlike the previous case where the Pareto front was a single line, in this case there are many lines, where each one represent the Pareto for a given base location. It can be seen that for each base location there is a limited efficiency possible with a given number of copters
(for example if the base is 2500 meters from the area the system can not reach more than 50 percent efficiency with 20 copters).

![Optimization Results with base location parameter](image)

**Fig. 12** Optimization results - With base variable

![Pareto front with base location parameter](image)

**Fig. 13** Pareto results - With base variable

**VI. Conclusions**

In this paper a SoS benchmark problem was introduced. The implementation of the problem in a generic modular AB-DES simulation framework was shown, while explaining the benefits of using this type of simulations for simulating complex system-of-systems. In addition, it was shown how the simulation can be coupled to an optimization environment and provide optimal solutions rapidly. The optimization results were analyzed to reveal optimal configurations and compositions for the benchmark problem. In the future the optimization will be upgraded to support uncertainties in order to examine its effect on the optimal configuration and composition.
Acknowledgments

The authors wish to acknowledge the kind support of Prof. Omri Rand from the Technion - Israel Institute of Technology and Dr. Robert Scott from the U.S. army. In addition, the support of Nir Zlotnik from Rafael is much appreciated.

References


