Natural Evolution Strategies for Task Allocation

Emir Besic, Jan Seeger*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: emir.besic@tum.de, seeger@in.tum.de

***Abstract*—In this paper we will be taking a look at Natural Evolution Strategies as a possible solution to the task allocation problem. Task Allocation appears in many areas of science, but is still missing a perfect solution. As the problem is NP-hard, it isn't possible to find a fast algorithm that always provides the optimal solution, which is why it is mostly solved using heuristic approaches. Natural Evolution Strategies is one such heuristic approach, showing promising results in the function optimization area. We take a look at the theory behind using it and compare it to different approaches, which are currently being used.**

***Index Terms*—task allocation, natural evolution strategies**

1. Introduction

Task allocation is a problem which occurs in many different forms, but the general form can be described as follows: There are tasks which need to be done, as well as agents which can do those tasks. For an agent to do a specific task he needs to pay a specific cost and he gets a specific profit. Each agent has a limited budget and he can only do as many tasks as he can pay for. The goal is to allocate the tasks to the agents, such that the profit is maximized and every agent stays within their budget.

One example is the factory allocation issue, which can be defined as follows. You can build a set amount of factories and need to allocate them to different locations. Each location has differing costs and profits associated with it. The factories would be the tasks and the locations would be the agents in this case. You can only place 1 factory at a location, so the budget of the agents would be 1 and each location offers a different profit due to a variety of reasons like tax and surroundings.

It is a very relevant problem lately, due to the rising popularity of distributed systems and IoT. Task allocation is a very common problem in these areas, so there is a need for good solutions.

There are some approaches which are still used and guarantee an optimal allocation, most prominent of which being the branch-and-bound algorithm. Jensen et al. in [1] describes the basics of this approach. They enumerate the candidate solutions in a special way, which enables them to search parts of the solution space only implicitly. In other words, it is possible to eliminate multiple candidate solutions just by checking one.

Unfortunately, finding the optimal allocation is NP-hard as proved by Cardellini et al. in [2], which means that it is not possible to find it in polynomial time. For this reason the problem is most commonly solved with heuristics. Heuristics may return sub-optimal solutions, but they are much faster than traditional approaches like the branch-and-bound solver.

There are many different heuristics which may be used for task allocation and we will be describing some of them in section 2. In this paper, we will be looking at a heuristic which shows a lot of promise, Natural Evolution Strategies (NES). NES is a state of the art function optimization algorithm which was first mentioned by Wiestra et al. in [3]. We will be describing NES in more detail in section 4. It isn't possible to directly apply NES for solving Task Allocation, which is why it needs to be adapted for this use-case. We will be taking a deeper look at why it is not suitable and a possible solution in section 5.

Another aspect is that there may be some optimization goals, which are specific to that instance of the problem. These goals tell the algorithm which solutions to prioritize. One such goal is optimizing energy usage as Seeger et al. describes in [4], where an allocation is considered optimal if it minimizes total energy use over an execution. Another optimization goal is response time and availability as Cardellini et al. describes in [2]. More optimization goals would be to minimize network usage, end-to-end latency, inter-node traffic etc. We will describe some use-cases in detail in section 2.

2. Background and Related Work

In this section we will be looking at some use-cases of task allocation.

Cardellini et al. considers the optimal operator placement for distributed stream processing applications in [2]. This is a basic use-case of the Task Allocation Problem. She has formulated the problem and implemented a solution which can be used as a benchmark against which to compare other placement algorithms.

Stanoi et al. looked at the problem of the distribution of query operators over a network of processors in [5] . He adapted a hill-climbing heuristics to reduce the search space of configurations.

Rizou et al. developed a distributed placement algorithm that minimizes bandwith-delay product of data streams between operators. He used a heuristic that first calculates the solution in an intermediate continuous search space and then mapping it to the physical network. ([6])

Gerkey et al. in [7] considers multi-robot task allocation (MRTA), which is also one of the textbook examples of our problem. He has formulated MRTA in a formal

doi: 10.2313/NET-2020-04-1_03

manner and given some commonly-employed and greedy solutions for the easier problems.

Seeger et al. tackles the problem of a malfunctioning IoT device, by allocating its tasks to other devices in the network in [4]. He solves the allocation problem by removing some constraints to transform it into a linear problem and using the simplex method [8].

In [9], Lewis et al. tackles the general problem by re-casting it into the form of an unconstrained quadratic binary program (UQP), which is then solved by a tabu search method developed for solving the general UQP model.

Cano et al. considers the problem of assigning software processes to hardware processors in distributed robotics environments in [10]. They model it as a task allocation problem and use constraint programming, a constructive greedy heuristic and a local search metaheuristic to solve it.

In [11], Wun-Hwa Chen et al. considers a special form of task allocation where they attempt to assign tasks to processors such that the communications cost among the processors, as well as the fixed costs of the processors are minimized. To solve this they use a hybrid method that combines Tabu search, described by Glover et al. in [12], for finding local optimal solutions and noising methods, described by Charon et al. in [13], for diversifying the search scheme.

Using evolution strategies for solving Task Allocation is not a new Idea, as it has already been discussed by Gupta et al. in [14]. The key difference is that they only consider some basic evolution strategies. But they already get some promising results, which gives us hope that NES might perform even better.

There have been innumerable other examples of Task Allocation, all with slightly different solutions. But it should be apparent now that this is a common problem without a commonly accepted perfect solution.

3. Modeling the Task Allocation Problem

We will base the model on the approach from Cardellini et al. in [2]. Let A denote the set of Agents and T the set of Tasks. Furthermore let P be a matrix such that the element $p_{\mathrm{i,j}}$ represents the profit when agent i does task j, let C be a matrix such that the element $c_{\mathrm{i,j}}$ represents the cost of agent i doing task j and let B be a set which contains the budget information of each agent such that agent i has budget b_{i}. There is another overview of the described parameters in table 1.

Symbol	Description
a_{i}	Agent with the index i
t_{i}	Task with the index i
$p_{\mathrm{i,j}}$	Profit when agent i does task j
$c_{\mathrm{i,j}}$	Cost when agent i does task j
b_{i}	Budget of agent i
$x_{\mathrm{i,j}}$	Represents if agent i was assigned task j

(1)

With these definitions we can now define Solving the Task Allocation problem for n tasks and m agents as the process of maximizing

$$\sum_{i=0}^{m}\sum_{j=0}^{n} p_{\mathrm{i,j}} x_{\mathrm{i,j}} \tag{2}$$

while also staying withing the budget for each i:

$$\sum_{j=0}^{n} c_{\mathrm{i,j}} x_{\mathrm{i,j}} \leq b_{\mathrm{i}} \tag{3}$$

Where $x_{\mathrm{i,j}} = 1$ when task j was allocated to agent i and 0 otherwise. Another constraint is that a task can only be assigned to a single agent which means that

$$\sum_{i=0}^{m} x_{\mathrm{i,j}} = 1 \tag{4}$$

for each j. In most instances of this problem there would be more constraints, but for the sake of simplicity, these will be the only constraints we will consider as they appear in every instance of the problem. This is a common formulation of the problem and it can also be used with a branch-and-bound solver in order to find the optimal allocation.

4. Natural Evolution Strategies

Function optimization problems appear in a variety of different scientific areas. Often these problems are not feasibly solvable in a short amount of time. Thankfully, small errors can sometimes be tolerated and the solution does not need to be optimal. This is where a heuristic approach like Natural Evolution Strategies (NES) ([3] and [15]) comes into play. They can find a solution to the problem in a much shorter time. The solution they come up with however, may not always be optimal, which is why it's important to pick the right heuristic. NES is one such algorithm which uses an heuristic approach for performing 'black box' function optimization. The structure of this function, also known as the fitness function, is unknown, but some measurements, chosen by the algorithm, are available.

4.1. How NES Works

To understand how NES functions, we need to first understand how the basic Evolution Strategies (ES) work. They are named as such due to their inspiration from natural Darwinian evolution. The basic idea is to produce consecutive generations of samples (candidate solutions) with each generation coming closer to the optimal result. We initialize the algorithm with a set of samples. They are then evaluated using the fitness function. The ones with the best performance are then chosen to create the next generation by *mutating* their genes, while the others are discarded. The Process is continued until a satisfying result is reached. This approach was proven to be very powerful, but it does have many problems. The most prominent being the high sensitivity to local optima (suboptimal solutions) and the slow speed of the evolution.

The Covariance Matrix Adaptation (CMA) algorithm is a much more sophisticated evolution strategy. CMA

doi: 10.2313/NET-2020-04-1_03

does not discard bad samples, but uses them to generate correlated mutations, which substantially speeds up evolution. It uses a multivariate normal distribution to draw mutations for the next generation. CMA is a major improvement to the previous algorithm, but it has an unpredictable nature and is still somewhat sensitive to local optima.

Natural Evolution Strategies keep the correlated mutations of CMA, but also try to reduce the sensitivity to local optima. NES estimates a gradient towards better expected fitness in every generation using a Monte Carlo approximation. This gradient is then used to update both the parent individual's parameters and the mutation matrix. NES uses a natural gradient instead of a regular gradient to prevent early convergence to local optima, while also ensuring large update steps. These differences make NES faster and less sensitive to sub optimal solutions compared to CMA.

4.2. Canonical NES

Now that we understand the concept of NES, we can take a look at a basic form of the algorithm.

Input: f, θ_{init}
repeat
  for $k = 1..\lambda$ **do**
    draw sample $z_k \sim \pi(\cdot|\theta)$
    evaluate the fitness $f(z_k)$
    calculate log-derivatives $\nabla_\theta \log\pi(z_k|\theta)$
  end
  $\nabla_\theta J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_\theta \log\pi(z_k|\theta) \cdot f(z_k|\theta)$
  $F \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_\theta \log\pi(z_k|\theta)\ \nabla_\theta \log\pi(z_k|\theta)^T$
  $\theta \leftarrow \theta + \eta \cdot F^{-1} \nabla_\theta J$
until *stopping criterion is met*;

Algorithm 1: Canonical Natural Evolution Strategies

Algorithm 1 was taken from [15] and depicts a pseudo code for the canonical NES algorithm.

The goal is to compute a gradient over the fitness function with regards to the search distribution θ and use it to update the distribution parameters, which are then used to draw the next generation of samples.

First of all, the inputs are the fitness function (f) and the initial parameters for the distribution (θ). Since NES usually uses the normal distribution, the parameters will be the mean (μ) and the standard deviation (σ).

The first step in the algorithm is to draw all the samples from the normal distribution. In order to do that, we need to evaluate the fitness of each sample and calculate their log-derivatives.

Once we have drawn and evaluated the desired amount of samples (λ) as well as calculated their log-derivatives, we can use that information to calculate the gradient and update the distribution parameters accordingly. These parameters will be used to create the next generation.

Now the only thing that is left is to repeat all the steps until a stopping criterion is met, or in other words, until we have a satisfying solution. This is the basic idea behind NES. In order to use it for task allocation, we will need to make some adjustments.

5. Solving Task Allocation with NES

As described in section 3, solving the Task Allocation Problem, is equivalent to optimizing a function, while staying within specific constraints. This is why using a state of the art function optimization algorithm like NES is a good approach. Unfortunately there is a problem with using the regular NES. It uses a multivariate normal distribution to draw samples, which is a continuous distribution and as such makes the algorithm incompatible with discrete variables. As our formulation from section 3 accepts only discrete solutions, we will need to adjust the algorithm accordingly.

In order to solve this problem, we will use the approach by Benhamou et al. in [16]. They found a way to make CMA compatible with discrete variables. They show in great detail that it is possible to extend the method for drawing samples to multivariate binomial correlated distributions, which are shown to share similar features to the multivariate normal used by CMA and NES. In other words, they show that the multivariate binomial distribution is the discrete counterpart of the multivariate normal distribution.

As described in section 4.1, NES is very similar to CMA. In particular, both use a multivariate normal distribution. The other differences like the natural gradient don't affect this method. So all we have to do is change our algorithm such that mutations are not drawn from a normal distribution, but a multivariate binomial one:

$$\mu + \mathcal{B}(\sigma^2 C) \tag{5}$$

With μ as the mean, σ as the standard deviation and C as the covariance matrix in the case of CMA-ES, which we will translate to NES.

Now that we have all the puzzle pieces we can put them together and define a basic algorithm. The algorithm itself is very similar to Algorithm 1, but there are some key differences.

Input: f, c, μ_{init}, σ_{init}
repeat
  for $k = 1..\lambda$ **do**
    draw sample $z_k \sim \mu + \mathcal{B}(\sigma^2 F)$
    evaluate the fitness
    check constraints c
    calculate log-derivatives
  end
  compute gradients
  update F
  update distribution parameters (μ, σ)
until *stopping criterion is met*;

Algorithm 2: Task Allocation with NES

Algorithm 2 shows a pseudo code for solving task allocation with NES. As it can be seen, the major differences to Algorithm 1 are first of all the inputs. To solve Task Allocation we need to pass our fitness function (2), but also the constraints (3) and (4). This is necessary as not all valid solutions may satisfy the constraints set by our model. This is also the reason why we need to check the constraints for each sample that we test. Another big difference is the distribution, from which we draw our

doi: 10.2313/NET-2020-04-1_03

samples. We are using the binomial distribution for the reasons mentioned before. The other difference is that we left out the exact calculations. They can be taken over from the original algorithm for the most part, but may need some small tweaks, since we are using a binomial distribution and there may be some room for improvement in the algorithm. In the original paper [15] there are already some mentions of the algorithm being tailored to specific use-cases (see sNES in section 3.5). To recognize whether there is room for improvement in our use-case it is necessary to implement and test the algorithm in a common task allocation scenario.

6. Conclusion

We have seen that Task Allocation is a widespread problem. There have been many different approaches to solving it, but due to it being NP-hard, it is very hard to find a suitable one. Most instances of the problem are solved either by the branch-and-bound approach, if the optimal solution is needed, or by heuristics, if smaller deviations from the optimal solution can be tolerated. These heuristics are often tailored to the specific instance of the problem and do not translate well into other instances. In other words, there still does not exist a perfect solution, which solves every instance of the problem optimally. Although we did not test our method, we can take a look at the results in the original NES paper [3]. They have tested NES on many different optimization problems and concluded that, NES can go toe to toe with most other function optimization algorithms. This and the fact that it has a polynomial complexity leads us to believe that it can be as good as, if not better than, currently used algorithms for some instances of the task allocation problem. Naturally, it is necessary to implement the algorithm first, before coming to any further conclusions, but what we can say, is that it is certainly an approach which is worth considering.

7. Future Work

The plan for the future is to first implement the algorithm and test how well it performs. It will almost certainly be necessary to tweak the calculations compared to Algorithm 1, in order to truly optimize it for task allocation. Once the algorithm is implemented and optimized for task allocation, we are confident that it will be a solution which offers both quality and speed. If NES turns out to be as good as we hope, it may be possible to find even more areas where NES could bring an improvement. Task Allocation isn't the only problem, which can be boiled down to a function optimization problem and is usually solved with heuristics. There are innumerable others, which is why the need for quality heuristic approaches is staggeringly big. So the next step after testing NES with Task Allocation is to find other similar problems in need of a better heuristic solution.

References

[1] J. Clausen, "Branch and bound algorithms – principles and examples," 1999.

[2] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli, "Optimal operator placement for distributed stream processing applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '16. New York, NY, USA: ACM, 2016, pp. 69–80. [Online]. Available: http://doi.acm.org/10.1145/2933267.2933312

[3] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Natural evolution strategies," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 3381–3387.

[4] J. Seeger, A. Bröring, and G. Carle, "Optimally self-healing iot choreographies," 2019.

[5] I. Stanoi, G. Mihaila, C. Lang, and T. Palpanas, "Whitewater: Distributed processing of fast streams," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, pp. 1214–1226, 10 2007.

[6] S. Rizou, F. Dürr, and K. Rothermel, "Solving the multi-operator placement problem in large-scale operator networks," *2010 Proceedings of 19th International Conference on Computer Communications and Networks*, pp. 1–6, 2010.

[7] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004. [Online]. Available: https://doi.org/10.1177/0278364904045564

[8] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 01 1965. [Online]. Available: https://doi.org/10.1093/comjnl/7.4.308

[9] M. Lewis, B. Alidaee, and G. Kochenberger, "Modeling and solving the task allocation problem as an unconstrained quadratic binary program," 04 2004.

[10] J. Cano, D. R. White, A. Bordallo, C. McCreesh, A. L. Michala, J. Singer, and V. Nagarajan, "Solving the task variant allocation problem in distributed robotics," *Autonomous Robots*, vol. 42, no. 7, pp. 1477–1495, Oct 2018. [Online]. Available: https://doi.org/10.1007/s10514-018-9742-5

[11] W.-H. Chen and C.-S. Lin, "A hybrid heuristic to solve a task allocation problem," *Computers & Operations Research*, vol. 27, no. 3, pp. 287 – 303, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054899000453

[12] F. Glover, "Tabu search—part i," *ORSA Journal on Computing*, vol. 1, no. 3, pp. 190–206, 1989. [Online]. Available: https://doi.org/10.1287/ijoc.1.3.190

[13] I. Charon and O. Hudry, "The noising method: a new method for combinatorial optimization," *Operations Research Letters*, vol. 14, no. 3, pp. 133 – 137, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016763779390023A

[14] A. K. Gupta and G. W. Greenwood, "Static task allocation using (μ, λ) evolutionary strategies," *Information Sciences*, vol. 94, no. 1, pp. 141 – 150, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0020025596000126

[15] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *Journal of Machine Learning Research*, vol. 15, pp. 949–980, 2014. [Online]. Available: http://jmlr.org/papers/v15/wierstra14a.html

[16] E. Benhamou, J. Atif, R. Laraki, and A. Auger, "A discrete version of CMA-ES," *CoRR*, vol. abs/1812.11859, 2018. [Online]. Available: http://arxiv.org/abs/1812.11859