Application of the “Alliance Algorithm” to Energy Constrained Gait Optimization

Valerio Lattarulo and Sander G. van Dijk

Adaptive Systems Research Group
University of Hertfordshire
Hatfield, UK

Abstract. This paper deals with the problem of energy constrained gait optimization for bipedal walking. We present a solution to this problem obtained by applying a recently introduced heuristic method, the Alliance Algorithm (AA), and compare its performance against a Genetic Algorithm (GA). We show experimentally that the intrinsic ability of the AA to handle hard constraints enables it to find solutions significantly better than the GA. Also with the constraint removed the AA show more reliable optimization results. Finally, we show that the final gait obtained through this method outperforms most solutions to this problem presented in previous works, in terms of walking speed.

1 Introduction

Despite a large amount of literature on the topic, general walking gait generation for bipedal robots is still very much an open problem. It is a hard problem due to the high dimensionality of a typical bipedal robot with many joints, the complexity of the dynamics of the system, and the difficulty of creating an accurate model that could enable supervised learning; given an arbitrary bipedal robot it is not obvious what the specifics of the motion of a good walking gait look like, one can often only formulate requirements that such a walk should meet.

One of the most important of these requirements is *stability.* Physical models of the walker’s dynamics can be used to solve this problem, usually done using methods based on stability concepts such as the zero-moment point (ZMP) [13] or zero rate of angular momentum (ZRAM)[3]. But in practice it can be difficult to derive such a model, and gaits generated with these methods generally do not meet another important requirement: *speed.* We apply the gait generation problem to the scenario of competitive robotic football, where speed is a major decider. Finally, in real systems a third requirement is *energy efficiency*; a fast, stable walk is not useful when it depletes the walker’s energy source before the end of its task. In this paper we will handle all three of these requirements.

Besides the already mentioned ZMP and ZRAM based methods, a range of other classes of gait generators have been put forward to solve this problem, such as central pattern generators (CPG) [5], and ones based on Fourier series [14], to which the generator used here is related. The methods in these classes

Table 1. Gait generation parameters

Parameter		Range	Unit
T	period	$[0-1]$	(s)
α_{up}	max up/down amp	$[0-40]$	(deg)
α_{fw}	max forward/backward amp	$[0-70]$	(deg)
α_{side}	max sideway amp	$[0-25]$	(deg)
α_{turn}	max turn amp	$[0-40]$	(deg)
α_{lean}	max lean amp	$[0-30]$	(deg)
θ_{vel}	bend velocity factor	$[0-45]$	(deg)
θ_{acc}	bend acceleration factor	$[0-40]$	(deg)
θ_{EMA}	EMA filter coefficient	$[0-1]$	

have different levels of complexity, biological plausibility and required levels of knowledge about the walker's dynamics. However, what they all have in common is that they rely on a set of parameters for which optimal values have to be found. Thus, before deciding upon a gait generating method, it is important to have good optimization methods available. In this paper we will contribute to this by applying the recently introduced Alliance Algorithm (AA)[1] to the problem of finding good parameter settings for a gait generator.

In the following section we will discuss in more detail the bipedal walking problem and the specific gait generator that we use. In section 3 we will describe the Alliance Algorithm. Next, we discuss how this method relates to popular other optimization methods. Section 5 will present the experiments performed to test the performance of the AA, and the results obtained. Finally, we will discuss these findings in section 6.

2 Problem Formulation

The problem that we address here is to find an optimal solution for a walking behavior for a robot in a simulated environment (Spark/RCSSServer3D [9]). Our gait generator is similar to the one described by Morimoto et al [8], in that it works by combining several basic oscillatory movements: 1) move feet up/down, 2) forward/backward, 3) left/right, 4) turn feet, and 5) lean left/right. These basic movements are created by sinusoidal oscillations of the joints, and walking is achieved by selecting the right phase for each and summing the basic motions, weighted relatively depending on the desired walking speed and direction. The phase-locking method to increase lateral stability used by Morimoto et al is not used here; in the scenario considered here frontal stability is a bigger issue. As a basic method to decrease this stability, the torso is bent backward or forward by a certain angle, based on a linear function of the desired walking speed and acceleration. Finally, an exponential moving average (EMA) filter is applied to smoothen changes in direction given by higher level behaviors. This results in the 9 parameters listed in table 1, the collection of which we denote with $\boldsymbol{\theta}$.

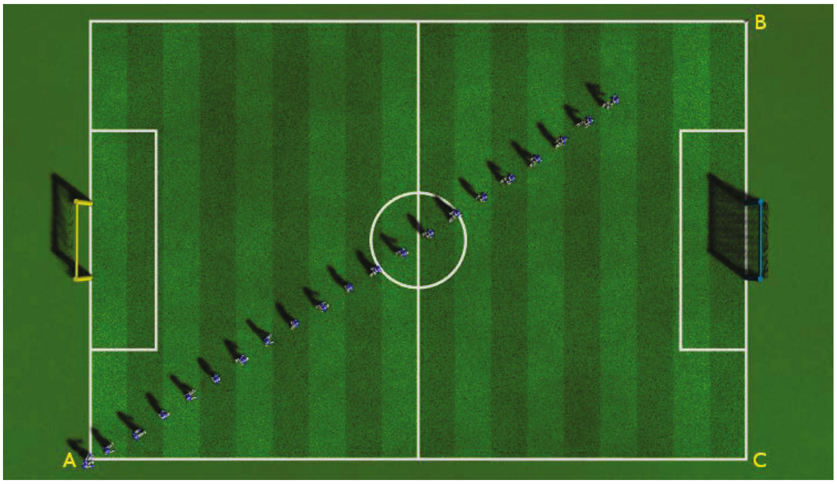

Fig. 1. Example trajectory

The problem then consists of optimizing these parameters in order to maximize the walking speed. As a proxy for this we use an objective function $F(\boldsymbol{\theta})$ that measures the final distance to a target point after an episode of a fixed amount of time, where a smaller distance is better. This choice was made over a function measuring simply the distance traversed in the episode since this could result in a gait that walks fast, but in the wrong direction. The chosen objective function also captures the requirement of stability, since an unstable gait that will make the robot fall will get less far.

In particular, the objective function measures the distance to a target point after an episode of fixed time. Figure 1 shows the trajectory of a sample run. The robot is placed in the bottom left corner of the simulated football field (marked 'A') and aims to walk to the upper right corner (marked 'B'), 21.6m away. It initially faces the *bottom* right corner (marked 'C'), to ensure that stable and fast turning is included in the optimization process. A fixed higher level system determines the walking and turning speeds at each time step, the task of the optimization system is to find the best parameters given this input. A single run consists of 20 seconds of simulation time. Further details of the experimental setup are given in section 5.

As mentioned in the introduction, speed and stability are not the only requirements; another important consideration is energy efficiency. Instead of including the energy use into the objective function, our methods allow us to explicitly pose an a-priori, fixed upper bound. This important benefit removes the necessity of elaborate calibration of the objective function to achieve the correct trade-off.

The energy E_j used by a joint j during time step t for some parameter assignment $\boldsymbol{\theta}$ is determined by:

$$E_{j,t}(\boldsymbol{\theta}) = P_{j,t}(\boldsymbol{\theta})\Delta t = (\tau_{j,t}(\boldsymbol{\theta}) \cdot \omega_{j,t}(\boldsymbol{\theta}))\Delta t, \tag{1}$$

where P is power in Watt, Δt is the length of time step t, τ is torque in Newton meter, and ω is angular velocity in radians per second. These latter three quantities can be obtained from the simulation, and summing the energy values over all joints and all time steps results in the total energy consumption as a function of the chosen parameters: $E_{tot}(\boldsymbol{\theta}) = \sum_{j,t} E_{j,t}(\boldsymbol{\theta})$.

Finally, we can formulate the problem to be solved as:

$$\max_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) \text{ subject to } E_{tot}(\boldsymbol{\theta}) \leq T, \tag{2}$$

where T is a certain fixed threshold. One of the main contributions of the work presented here is the inclusion of a hard constraint, though for completeness we will also handle the unconstrained version.

3 The Alliance Algorithm

The Alliance Algorithm (AA) [1] is based on the metaphoric idea that a certain number of tribes struggle to conquer an environment, which offers resources to survive. Two features characterize each tribe: its strength, and the resources necessary for its survival. In order to increase their strength, the tribes may enter into alliances. This alliance then is characterized by a new strength and amount of necessary resources, dependent on the features of the tribes inside the alliance. The AA proceeds by forming alliances that will be more preferable as their strength is higher than other alliances, but do not consume more than the maximum available amount of resources. The algorithm will end when the strongest alliance is created: it will be able to force out other alliances and conquer the environment, while still meeting its resource constraints. The combination of the tribes that compose the strongest alliance represent the solution of the problem. After convergence it is possible to perform another iteration of the algorithm, starting with tribes that are influenced by the strongest alliance of the previous iteration, to further refine the solution.

Given a certain problem with a certain solution space:

- A single tribe t is composed of the tuple $(\boldsymbol{\theta}_t, s_t, r_t, a_t)$, i.e. a point in solution space $\boldsymbol{\theta}_a$, a strength s_t, a resource demand r_t, and an alliance assignment a_t.
- The set of alliances is a mutually disjoint partition of tribes. Each alliance a again represents a point $\boldsymbol{\theta}_a$ in the solution space, defined by the tribes that compose the alliance and a problem specific joint function.
- The strength s_i of tribe or an alliance is the value obtained with the objective function on the respective solution:

$$s_i = F(\boldsymbol{\theta}_i). \tag{3}$$

- The amount of resources needed by a tribe or an alliance is the value obtained by a constraint function on the respective solution:

$$r_i = E(\boldsymbol{\theta}_i) \tag{4}$$

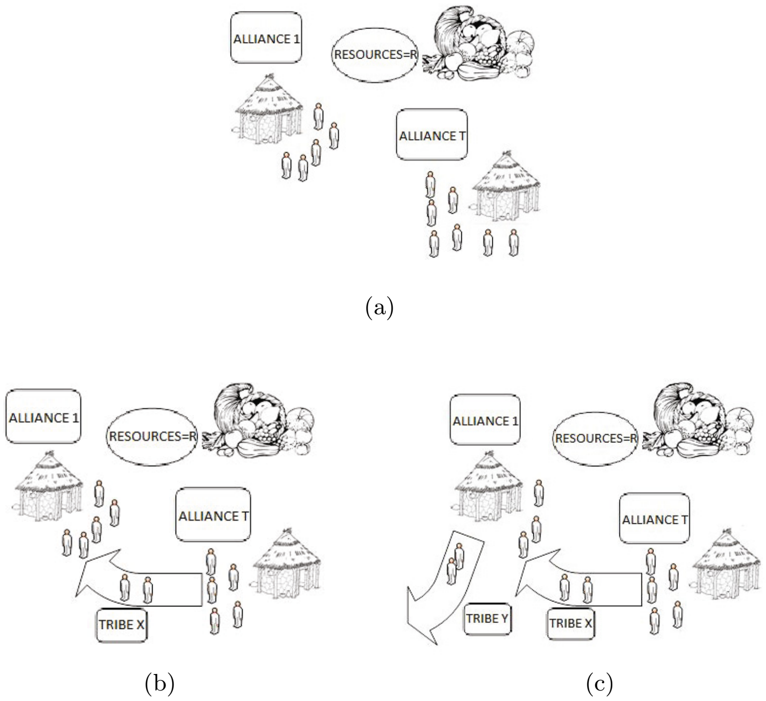

Fig. 2. Dynamics of the Alliance Algorithm. a) Tribes form alliances in an environment with limited resources. b) A tribe can join another alliance. c) By doing so, an inferior tribe can be removed from the alliance.

Initially, N tribes are generated, where the assignments of $\boldsymbol{\theta}$ for each tribe can be taken at random, as done in the experiments presented here, or in a problem specific way when higher level prior knowledge is available. Each is assigned a unique ID $t \in [1-N]$, and its strength s_t and resource demands r_t are determined using $F(\boldsymbol{\theta}_t)$ and $E(\boldsymbol{\theta}_t)$. These properties of the tribes are now fixed; as is shown below, movement through solution space during execution of the algorithm is accomplished by the metaphoric aggregation process, i.e. the formation of the alliances and by doing so the combination of tribes, accomplishes movement in the solution space. To finalize the initialization, each tribe is assigned to its own alliance, such that $a_t = t$. After this point, the ID of an alliance will always be set to the lowest ID of its member tries. This is done to guarantee that each alliance us uniquely identified and to maintain a necessary ordering. Note that an alliance can become empty if the last tribe in that alliance joins an alliance that already contains a tribe with a lower ID. In the remainder of the text we will treat single tribes as ‘alliances of one’ where applicable.

After initialization, the algorithm starts to iteratively search for the best possible alliance. To do so, at each step first a token is given to an alliance (in our experiments randomly, but again this can be adapted with a different token function), which can now ask a tribe outside of the alliance if it wants to join this alliance. The method of selection a tribe to ask can again be implemented differently for different problems, here random selection is used. However, the alliance receiving the token can choose only tribes that it did not ask before. The request of the alliance to the chosen tribe can have one of several outcomes, dependent on a given decision function.

To exemplify, assume the generic case shown in Fig.2(a), where the tribes of the first alliance are:

$$\Big[(s_1, r_1, a_1), \ldots, (s_p, r_p, a_1)\Big],$$

and that the tribe that received the request, tribe x, is part of an alliance containing the following tribes:

$$\Big[(s_t, r_t, a_t), \ldots, (s_N, r_N, a_t)\Big],$$

where 1 is the minimum of the IDs of the tribes in the first alliance, t of the tribes in the second, and $x > t$.

The possible responses (used for our purposes) to the request are:

1. Tribe x can join the first alliance, and by doing so abandon the second alliance, as shown in Fig. 2(b). This results in the two alliances now containing:

$$\Big[(s_1, r_1, a_1), \ldots, (s_x, r_x, a_1), \ldots, (s_p, r_p, a_1)\Big], \Big[(s_t, r_t, a_t), \ldots, (s_N, r_N, a_t)\Big]$$

2. Tribe x can join the first, and by doing so abandon the second alliance, and *replaces* an inferior tribe y in this first alliance, as shown in Fig. 2(c). This now results in three alliances:

$$\Big[(s_1, r_1, a_1), \ldots, (s_x, r_x, a_1), \ldots, (s_p, r_p, a_1)\Big], \Big[(s_t, r_t, a_t), \ldots, (s_N, r_N, a_t)\Big],$$
$$\Big[(s_y, r_y, a_y)\Big]$$

3. Tribe x does not join the new alliance when the resulting new alliance is not stronger than the tribe's current alliance, or when the resource consumption of the resulting new alliance is to high.

After that has been decided an action, the effective movement of the tribes and/or alliances and an update of all the data structures is performed, and another cycle is started with the assignment of the token to another alliance. In this way, alliance are formed and broken iteratively, until convergence. This happens when all alliances have asked all other tribes to join without success, and therefore no changes are made. Now, the strengths of all final alliances are compared, and the alliance that has the highest strength without exceeding the available resources wins. The point in solution space $\boldsymbol{\theta}_a$ represented by this alliance is the final solution of the problem. Another iteration of the algorithm can be started, seeded with new tribes that can have characteristics similar to the previous strongest alliance, as chosen by the initialization function.

There are several parameter values and functions that have to be chosen when using this algorithm. For the particular problem handled in this paper, a tribe is composed of values for the 9 continuous gait generation variables, within their range as listed in table 1. The initialization function uses uniform random values within this range at in the first iteration, but in subsequent iterations, the values are influenced by the best solution of the previous iteration. Every time that the algorithm restarts, the section of the solution space from which tribes are sampled is centered around the currently known best solution and reduces 5% in size compared to the section used in the previous iteration. A tribe is only allowed to join another alliance only if the passage is convenient for both the entities (in terms of strength and resources) involved in the transition, otherwise the tribe will not join the new alliance. Finally, as mentioned before, the solution of a particular alliance is a function of its tribes, which means that the single tribes are placed in a particular area of the solution place, but when they join together the resulting alliance is placed in another area according to some combination function. For our purposes, this combination is an averaging over the parameter values of the tribes.

4 Related Methods

Similar to a Genetic Algorithm (GA) [4] where individuals can be combined, e.g. through cross-over, the AA moves through solution space by merging solutions. The important difference however, is that in the AA the resulting solutions can be 'broken up' again, when the combination did not turn out to be beneficial in the end. In the AA's pure form, combination is the only search operation, which means that an initial seeding with N tribes defines a set of $\binom{N}{1} + \binom{N}{2} + \cdots + \binom{N}{N}$ possible alliances that the algorithm will search through. However, as we discussed it is possible to perform multiple iterations, where the tribes in a new iteration are generated to be similar to the winning alliance of the last, in order to increase the search space. When doing so, as we have shown, one can also reduce the size of the section of solution space from which new tribes are sampled, to 'home-in' on and refine the best solution, analogous to gradually decreasing the temperature in Simulated Annealing [7].

In the AA, tribes work together to find a better solution, something that is also accomplished, in different ways, by some other optimization algorithms, such as Ant Colony Optimization (ACO) [6] and Particle Swarm Optimization (PSO) [2]. However, in the AA every tribe is essentially selfish and joins with other tribes only if there is an advantage to itself. There is no fixed 'social contract', and every separate tribe and alliance will take any opportunity to conquer the environment. Moreover, in ACO and PSO the individuals move through solution space according to the set of rules of the respective algorithm, whereas in the AA the tribes themselves do not have the tendency towards going near the best solution. Instead, they can join together to form an alliance that will be placed in another area of the solution space, possibly far from the solution represented by the individual tribes.

The selfish behavior of the tribes make the AA also related to greedy search (used in e.g. [11] for the same problem as presented here), but, again, the option of a tribe to leave an alliance directly creates the possibility of backtracking and trying alternative combinations.

Another popular method showing overlap with the AA is the Covariance Matrix Adaptation-Evolution Strategy (CMA-ES). This is an optimization algorithm based on updating the covariance matrix of a distribution over solutions in order to make the sample previously successful search steps more likely. In this algorithm, new samples are combined through weighted averaging to arrive at a new mean of the distribution. This is similar to how we will form alliances by averaging the solutions of their tribes. However, in the AA one can choose any other recombination method, whereas averaging is a fixed integral part of the CMA-ES. Moreover, the mean of all the sampled solutions as used in the CMA-ES is a particular case of the AA in which all the tribes ally together in a single big alliance. This normally however does not happen, as alliances do not accept detrimental solutions, and, in contrast to the CMA-ES, the AA can break up combinations again at a later stage if that turns out to be beneficial.

Finally, the AA aggregates in a way similar to the NEAT algorithm [12], which is a method used for growing the topology of a neural network. The difference with the AA is that there is only one topology that evolves and could increase indefinitely in size, but in the AA there are several alliances that can interact each other and their size is limited by the initial set of tribes.

5 Experiments

In this section we will present the experiments performed to test the performance of AA on the gait generation problem. Besides obtaining pure performance measurements for AA, we will also present a comparison of AA with a GA. This was chosen as the most important comparison method, because it is well understood, and has been used successfully for the purposes presented here, by the authors as well as by others [10]. One of the main benefits of AA is the straightforward way of including hard resource limitations. However, to our knowledge no previous work including such constraints has been performed in the scenario used here, so for completeness and to be able to compare better with related work, we have also performed experiments without energy constraints.

The parameters used for the AA are: number of tribes $N = 10$, and the size of the solution space is reduced 5% after each iteration. For further choices of the different functions of the algorithm, see section section 3.

As mentioned before, a Genetic Algorithm (GA) is used as a reference. Here, each individual consists of a single instantiation of the 9 walking parameters, where each parameter forms a gene. During evolution a population size of 60 is used, with a cross-over rate of 0.1, and a mutation rate on each gene of 0.1. Tournament selection with tournament size 5 is used as the selection method.

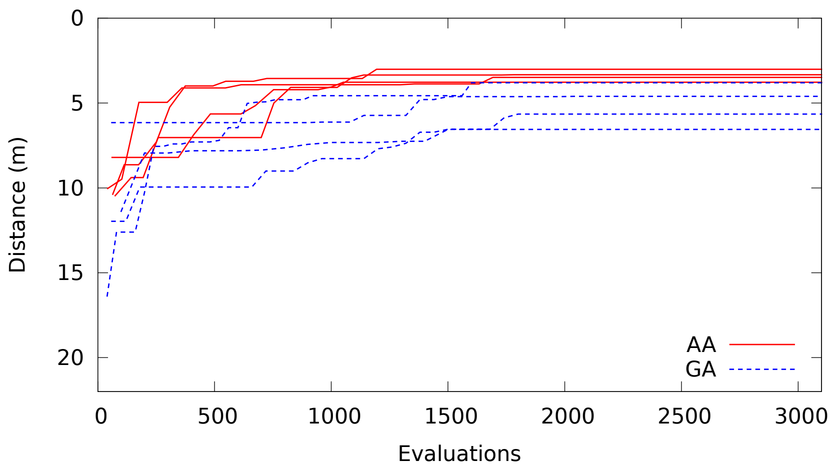

Fig. 3. Best solution found, against cumulative amount of evaluations performed, without energy constraint. AA = Alliance Algorithm, GA = Genetic Algorithm.

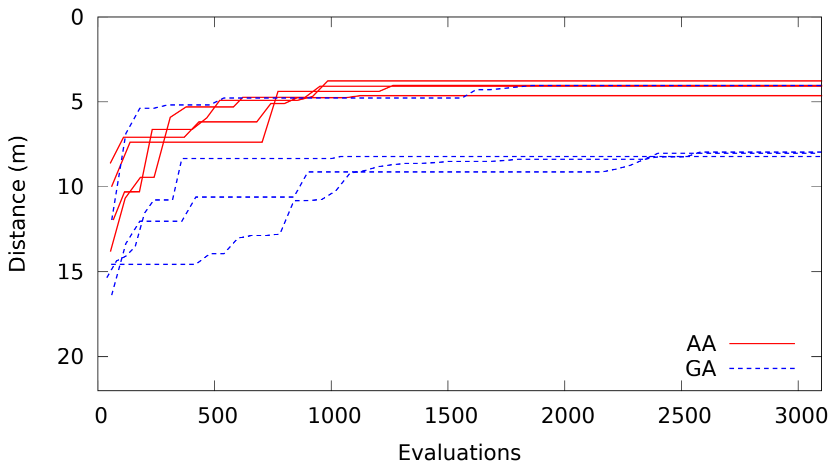

Fig. 4. Best solution found, against cumulative amount of evaluations performed, with energy constraint. AA = Alliance Algorithm, GA = Genetic Algorithm.

Firstly, we have performed 4 unconstrained experiments with the AA and 4 with the GA, and have recorded the best solutions after each iteration/generation. These values are plotted in Fig. 3 against the cumulative number of evaluations of the objective function, i.e. the total number of runs of the scenario of Fig. 1. Note that this number is fixed for each generation of the GA (i.e. 60), but that the number of evaluations for a single iteration of the AA is not fixed, and in our experiments fluctuates around 60.

Secondly, we introduced the energy constraint. After analyzing the solutions found in the first set of experiments, a maximum total energy consumption of 150,000 Joule was chosen as a threshold in order to give a sufficient limitation without constraining the solution space too much. In the AA, this was readily incorporated as the amount of resources available in the environment. Applying such a constraint to a GA, or other similar methods, is less straightforward and can require additional bootstrapping experimentation. In this case, we arrived at including the energy use in the objective function as a negative penalty, but only when this exceeds the maximum of 150,000 Joule. Due to the difference in scale of several orders of magnitude between distance traversed and energy use, this results in a semi-lexicographic selection that preserves a gradient outside of the constraint. However, it is important to note the relative arbitrariness of this choice compared to the AA. The development of the best solutions for these scenarios is shown in Fig. 4.

6 Discussion

The results presented in the previous section offer several insights into the usability of our methods. Firstly we can note that the fastest gait, found by the Alliance Algorithm, achieved an average speed of 0.93 m/s. As a compariosn, Shafii et al [10] have presented to application of PSO with a gait generator similar to, but less restricted than, the one we use, which obtains a speed of 0.77 m/s. Also compared to speeds obtained by some of the high ranking teams in the RoboCup 3D Soccer Simulation competitions, reproduced from [11] in table 2, our best gait scores well. Especially when one notes that the measurements of

Table 2. Comparison of the average speed in different teams

Team	Speed (m/s)
Proposed Approach	0.93
Shafii et al	0.77
BoldHearts	0.68
Wright Eagle	0.67
SEU	1.20
Bats	0.43

the other gaits have been made on a straight walk, whereas in our experiments the agent is presented with the additional handicap of having to first turn into the right direction.

When we look at Fig. 3, we see that the GA is able to find a solution with performance close to that found by the AA. However, this only happened in one out of the 4 performed experiments, where it was lucky to already have a good solution after the first generation, and still after close to twice the amount of evaluations of the objective function than the slowest run of the AA. In the other three experiments all final solutions were inferior. In contrast, each run of the AA was able to find a solution close in performance to the optimal one found, with half of the time doing so within just 500 evaluations.

In the main results of this paper, those of the experiments with energy constraints, this difference is increased further. To our surprise, both methods found solutions with speed not far from that of the best unconstrained solution, even though the solutions in the first set of experiments went well above the energy constraint. However, again the GA only happened on a solution of this quality once; the other three runs resulted in solutions that were much worse. The AA on the other hand again managed to find a good solution in each of the 4 runs. A GA that uses another method to incorporate the energy constraint may perform better. However, such trial-and-error calibration of the algorithm can consume a lot of time, especially in scenarios like the current one where many evaluations of the objective function are very costly. In contrast, the AA naturally accommodates for hard constraints such as the constraint on total power consumption.

Visually it is difficult to discern the qualitative difference between the efficient and inefficient solutions. However, analysis of the solution parameters indicates that in the constrained experiments gaits with lower frequencies are preferred, which results in energy saving thanks to lower angular rates and accelerations.

The results presented in this paper show that the AA can offer a fast and reliable method for finding solutions in complex control problems, such as the gait generation problem for bipedal robots.

Finally, we would like to point out the generality of the algorithm achieved by the possibility of applying any of a wide range of functions to combine and aggregate solutions in a natural way. Moreover, a tribe can represent anything a full solution, but also a partly solution. With this approach it is possible to combine together different entities that together constitute the final solution, such as different behaviors with their separate parameters that combine together into a higher level behavior.

References

1. Calderaro, V., Galdi, V., Lattarulo, V., Siano, P.: A new algorithm for steady state load-shedding strategy. In: 12th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM), pp. 48–53 (2010)
2. Colorni, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: European Conference on Artificial Life, pp. 134–142 (1991)
3. Goswami, A., Kallem, V.: Rate of change of angular momentum and balance maintenance of biped robots. In: ICRA, pp. 3785–3790 (2004)
4. Holland, J.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press, Cambridge (1992)
5. IJspeert, A.J.: Central pattern generators for locomotion control in animals and robots: A review. Neural Networks 21(4), 642–653 (2008)
6. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (August 1995)

7. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)
8. Morimoto, J., Endo, G., Nakanishi, J., Cheng, G.: A biologically inspired biped locomotion strategy for humanoid robots: Modulation of sinusoidal patterns by a coupled oscillator model. IEEE Transactions on Robotics 24(1), 185–191 (2008)
9. Obst, O., Rollmann, M.: SPARK – A Generic Simulator for Physical Multiagent Simulations. Computer Systems Science and Engineering 20(5), 347–356 (2005)
10. Shafii, N., Aslani, S., Nezami, O.M., Shiry, S.: Evolution of Biped Walking Using Truncated Fourier Series and Particle Swarm Optimization. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S. (eds.) RoboCup 2009. LNCS (LNAI), vol. 5949, pp. 344–354. Springer, Heidelberg (2010)
11. Shafii, N., Reis, L.P., Lau, N.: Biped Walking Using Coronal and Sagittal Movements Based on Truncated Fourier Series. In: Ruiz-del-Solar, J., Chown, E., Ploeger, P.G. (eds.) RoboCup 2010. LNCS (LNAI), vol. 6556, pp. 324–335. Springer, Heidelberg (2010)
12. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2), 99–127 (2002)
13. Vukobratovic, M., Borovac, B.: Zero-moment point-thirty five years of its life. International Journal of Humanoid Robotics 1(1), 157–173 (2004)
14. Yang, L., Chew, C., Poo, A., Zielinska, T.: Adjustable bipedal gait generation using genetic algorithm optimized fourier series formulation. In: IROS, pp. 4435–4440. IEEE (2006)