Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2017)
F. Jaillet and F. Zara (Editors)

Inverse Kinematic Solutions for Articulated Characters using Massively Parallel Architectures and Differential Evolutionary Algorithms

Ben Kenwright

Southampton Solent University†

Abstract

This paper presents a Differential Evolutionary (DE) algorithm for solving multi-objective kinematic problems (e.g., end-effector locations, centre-of-mass and comfort factors). Inverse kinematic problems in the context of character animation systems are one of the most challenging and important conundrums. The problems depend upon multiple geometric factors in addition to cosmetic and physical aspects. Further complications stem from the fact that there may be non or an infinite number of solutions to the problem (especially for highly redundant manipulator structures, such as, articulated characters). What is more, the problem is global and tightly coupled so small changes to individual link's impacts the overall solution. Our method focuses on generating approximate solutions for a range of inverse kinematic problems (for instance, positions, orientations and physical factors, like overall centre-of-mass location) using a Differential Evolutionary algorithm. The algorithm is flexible enough that it can be applied to a range of open ended problems including highly non-linear discontinuous systems with prioritisation. Importantly, evolutionary algorithms are typically renowned for taking considerable time to find a solution. We help reduce this burden by modifying the algorithm to run on a massively parallel architecture (like the GPU) using a CUDA-based framework. The computational model is evaluated using a variety of test cases to demonstrate the techniques viability (speed and ability to solve multi-objective problems). The modified parallel evolutionary solution helps reduce execution times compared to the serial DE, while also obtaining a solution within a specified margin of error (<1%).

Keywords: *inverse kinematics, animation, differential evolution, multi-threading, patterns, motion, detection, poses, articulated skeletons*

1 Introduction

Solving Inverse Kinematic (IK) problems efficiently and quickly is challenging and important. The challenges stem from the fact that inverse kinematic problems are typically highly non-linear and multi-modal, not to mention, requiring prioritization control (e.g., minimal change in angle/position, centre-of-mass location, obstacle avoidance and joint singularity circumvention). For instance, a particular element of articulated character structures often overlooked is the tightly-coupled nature of the problem (minor changes to joints lower in the hierarchy will influence inherited joints) [Ken12b], including the 'discontinuous' search space due to joint limits and conflicting goals which we address in this paper using an evolutionary algorithm.

Modelling programs (e.g., Blender [Ble17]) are limited to solving local optimisation (such as, end-effector placement location) and do accommodate global multi-objective problems (e.g., centre-of-mass and controlled stylistic input) due to the complexity of the search space. For instance, as the complexity of the search space increases due to the multiple objectives, the search space becomes more non-linear and discontinuous dramatically increasing search times and the chance of converging on an optimal solution. However, human-like characters possess a vast assortment of degrees of freedom (DOF) that allows them to perform a diverse range of actions. Controlling a character's pose to generate targeted actions that accomplish specific tasks (e.g., reaching and stepping) while controlling physical aspects of the pose is of particular interest [Ken12b, GMHP04] Many kinematic optimisation algorithms are only effective at finding 'local' optimal solutions (e.g., gradient-based algorithms like, steepest decent and hill-climbing). Global kinematic optimisation techniques are often required for 'multi-model' and 'highly non-linear' problems (e.g., DE algorithm). These kinematic optimisation problems are often

† benjamin.kenwright@solent.ac.uk

DOI: 10.2312/vriphys.20171085

extremely challenging (especially for problems with large numbers of parameters and fitness criteria). Of course, recent research into metaheuristic algorithms have suggested that population based optimisations are promising for solving these tough optimisation problems [Yan12].

As we explain in this paper, the heuristic Differential Evolution algorithm has three main advantages; (1) able to find an optimal minimum for a multi-modal search space regardless of the initial parameter values, (2) fast convergence, and (3) uses few control parameters. As we show, the DE algorithm is particularly well suited for numeric optimization problems (e.g., multi-object kinematic tasks). While the DE algorithm is a population based solution similar to a genetic algorithm, it does not require as many parameters to tune (mutation, selection, crossover).

Contribution The key contributions of this paper are: (1) we evaluate and demonstrate the practical viability of the differential evolutionary algorithm for multi-modal kinematic problems; (2) we amalgamate the differential evolutionary algorithm with a massively parallel execution environment to reduce the time-frame for solving complex multi-modal inverse kinematic problem; and (3) we evaluate and discuss the limitations and bottlenecks of the differential evolutionary algorithm in addition to comparing the sequential and parallel algorithm for solving non-linear inverse kinematic problems.

2 Related Work

2.1 Inverse Kinematics

Articulated structures are used to represent virtual character skeletons (e.g., humans, insects and horses). The IK problem comes down to the problem of finding the angles for the joints to achieve some expected goal (end-effector location, pose or obstacle avoidance). For simple IK character problems, such as, knee or arm locations with 2 or 3 degrees of freedom, we can use analytical methods to find a solution [Kal08]. However, for more complex IK systems, we require numerical methods due to the highly non-linear and discontinuous nature of the problem [TGB00]. Numerical methods are able to find an 'approximate' solutions (i.e., closest or best guess) through an iterative manner. However, these methods are vulnerable to local optimal and long convergence times. For example, the Cyclic Coordinate Descent (CCD) algorithm [Ken12a] is a computationally inexpensive technique based upon a hybrid heuristic technique based upon a iterative trigonometric search. The CCD algorithm is a good compromise between speed and accuracy but it limited end-effector problems (does not map to solving different multi-objective problems). On the other hand, evolutionary algorithms are another type of numerical method, such as, the genetic algorithm, which have the advantage of being able to search difficult search spaces (handle redundancy and multiple goals) as demonstrated by Parker et al. [PKG89] and Kallmann et al. [KAAT03]. The genetic algorithm has also been parallalized using the GPU for solving character animation problems [Ken14]. The method works towards minimizing a fitness criteria (e.g., minimal joint displacement) to find the optimal solution to the problem. While the Genetic Algorithm (GA) has a number of advantages and has shown promising results it does require the hand-tuning of multiple parameters to achieve acceptable performance (e.g., convergence speed). What is more, the parameters need to be re-tuned for each problem (i.e., the search space) [GC00]. The data also need to be mapped to a genome for cross-breeding - which can be unintuitive depending upon the problem. In this paper, we continue with an evolutionary paradigm but base it upon a 'differential' model to reduce the number of parameters and the representation of the data [SP97].

Various research has approached different problems in the context of inverse kinematic algorithms, such as, enforcing an arbitrary number of strict priority levels [BB04] to minimising the effort in redundant manipulators [DW97]. In particular, gradient-based IK methods are a popular solution for end-effector problems [KSB10, BK05]. Robotic humanoid with end-effector constraints for resolved motion rate control (RMRC) [DVS01] using a statistical learning algorithm. On the positive side, gradient based methods have the advantage of converging quickly on similar solutions, yet struggle with discontinuous or multiple objective problems. In comparison our approach is primarily a heurstic model with an underlying 'gradient' concept. This means the technique does not purely rely on 'gradient' information for local search optimisations. In fact, for complex search spaces with constraints and multiple fitness criteria gradient information may be non-existent, unreliable or poorly specified causing issues for convergence.

2.2 Evolutionary Algorithms

Genetic Algorithms (GA) Genetic algorithms keep pretty closely to the metaphor of genetic reproduction. While both genetic algorithms and differential evolution are examples of evolutionary computation they have distinct advantages and disadvantages depending upon the problem. For example, GAs operate by mimicking the language of biology, using chromosomes, genes, crossover and mutation to represent the problem. This translating of data to a low-level biological form can disconnect the optimisation problem from the goal and induce additional computational and managerial overhead [Deb00, Ken14]. One of the first people to apply GA to solving IK problems was Joey and David [PKG89] who solved redundant robotic problems using a 'single' fitness function. This later led onto numerous modified solutions, such as, the Multi Population Genetic Algorithm (MPGA) by Zhen and Yan-Tao [SJTJ15]. and an optimised a genetic algorithm for real-time inverse kinematic trajectories called the Dynamic Multi-layered Chromosome Crossover (DMCC) algorithm by Albert et al. [AKT*08].

Differential Evolution (DE) Differential evolution is in the same style as GA, but the correspondences are not as exact. The first big change is that DE is using actual real numbers (in the strict mathematical sense, while the technique is implemented as real numbers (e.g., floating or double precision values) enabling the technique (in theory) to cover the complete range of real number values. As a result, the ideas of mutation and crossover are substantially different. The mutation operator is modified and represented as a 'probabilistic' mixing of the population - to accomplish the same purpose of breaking things out of local minima [Pri96, QHS09].

On the plus side, there are a handful of results showing DEs are

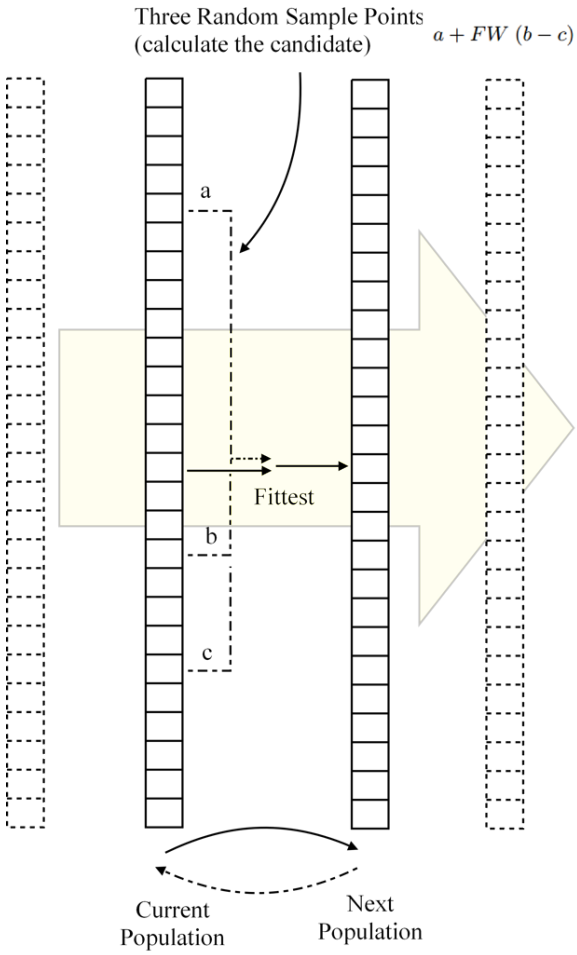

Figure 1: ***Parallel DE Implementation*** *- The parallel implementation calculates an candidate for each individual for the next population - if the new candidate has a greater fitness than the original individual it is replaced. Since we have the current and next memory locations, we do not have any 'blocking' calculations (i.e., neighbouring dependencies such as reading/writing to the same memory location).*

often more effective and/or more efficient than genetic algorithms [VT04, QHS09, OLMY14]. For instance, when working in numerical optimization, it is valuable to be able to represent things as actual 'real numbers' instead of having to work around representations, such as, chromosomal values. Importantly, the DE performs well for certain situations because the vectors can be considered to form a 'cloud' that explores the high value areas of the solution-space quite effectively. The concept closely mimics a particle swarm optimization in some senses [VT04]. On a side note, as DEs are based upon the evolutionary concept, they still have the usual problem of local minima depending upon the fitness functions causing bottlenecks and exasperating search times.

Recently the DE was employed in a robotic IK problem by Rodriguez et al. [RSRHO]. Rodriguez et al. [RSRHO] presented a

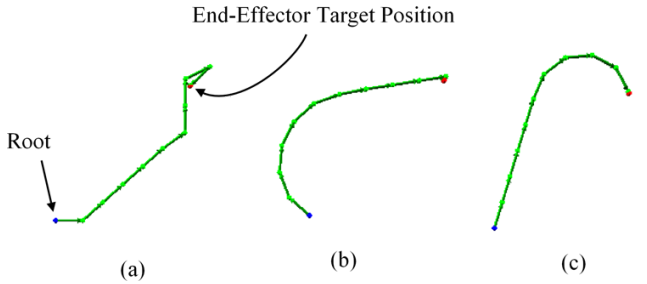

Figure 2: ***Multiple Fitness Criteria*** *- (a) Single objective end-effector (distance); (b) Multi-objective: end-effector distance plus a secondary biasing constraint for the links near the start of the chain to remain as straight as possible; and (c) Multi-objective: end-effector distance plus a secondary object to keep links near the end-of the chain as straight as possible.*

four Degrees of Freedom (DOF) solution for a robotic system using a Differential Evolution algorithm. Prior to that Gonzalez and Blanco [GBM12] showed the DE increased convergence in a memetic IK solution. A hybrid DE was proposed by Nguyen and Ho Pham [SAC14] to train an adaptive neural network for the solution of inverse kinematic (solve the inverse kinematic of a 3 DOF manipulator).

Our Work While the DE has gained popularity in robotics for solving IK problems, we are not aware of it being used in the graphical community. The paper presents a differential evolutionary (DE) approach for solving inverse kinematics (IK) problems (e.g., articulated character systems). The interest in using DE is the flexibility in searching the space for possible solutions. We reduce the search times through the adaptation of the algorithm onto the GPU.

3 Method

We combine the Inverse Kinematic problem with multiple tasks (e.g., minimal change in angle/position, centre-of-mass location, obstacle avoidance and joint singularity avoidance) to evaluate the DE algorithm. We demonstrate the viability of the DE optimisation method by solving complex kinematic problems in relation to convergence times, population sizes, and parameters. We start with simple single objective problems and extend them to include multi-objective problems in addition to prioritization using a weighted sum objective function. Examining the results from a sequential DE optimisation algorithm it provides a reference point for quantifying the quality of the modified parallel version. However, we explore and explain the problems and modifications necessary to port the DE technique to the graphical processing unit (GPU).

Population As shown in Figure 3, we can see that the DE optimisation algorithm is able to converge on a solution with a limited number of generations even for small populations (e.g., less than 100). What is more, Figure 3 shows, there is a relationship between population size and the number of generations required to find a solution. Hence, reducing any computational bottlenecks for rapidly advancing generations and the ability to instance large population sizes would seriously reduce computational times (i.e., paralleliza-

```
Data: Parameters & Initialization
// Differential Weight [0,2]
FW = 2
// Crossover Probability [0,1]
CR = 0.5
// Dimensions
N = Number of Joint Angles
// Population Size
POP_SIZE
// Array Individuals (Population)
Initialize to random values

// Evolution Function
Function Epoch()
    for (j=0; j<POP_SIZE; ++j) do
        // Pick a random candidate from population
        x = RandomNumber(0, POP_SIZE-1)

        // Pick three different random points
        // from population
        a = RandomNumber(0, POP_SIZE-1);
        b = RandomNumber(0, POP_SIZE-1);
        c = RandomNumber(0, POP_SIZE-1);
        // Note a b and c cannot be the same

        // Pick a random index [0-Dimensionality]
        R = RandomNumber(0, N-1);

        for (i=0; i<N; ++i) do
            if ( i==R || RandomNumber(0,1)<CR ) then
                candidate=a+FW*(b-c)
            end
        end

        // See if new individual is better than the original //
        candidate, if so replace
        if ( candidateFitness < originalFitness ) then
            population[x] = candidate;
        end
    end
    generation++
EndFunction
```

Algorithm 1: Sequential Algorithm for the Differential Evolutionary Implementation.

tion of the technique). The details of the algorithm for solving IK problems:

- The joint angles are stored as 'Euler' angles to reduce memory footprint for storing large populations (compared to matrices)
- Euler angles enable us to easily limit ranges (intuitive to artists)
- Forward kinematic calculations can be performed through the generation of the transform matrices (e.g., rotations/translations)
- During the forward kinematic update, we are able to calculate the fitness details (end-effector locations, difference between current and desired angles, location of the overall centre-of-mass)
- The individuals within the population have the joint angles randomly initialize to values between $-\pi$ and π during the initialization phase
- Since we focus on articulated character structures, the different types of joints (e.g., hinge and ball-joint) can all be represented using Euler angles with limits (i.e., only angular joints no sliders - rigid lengths)

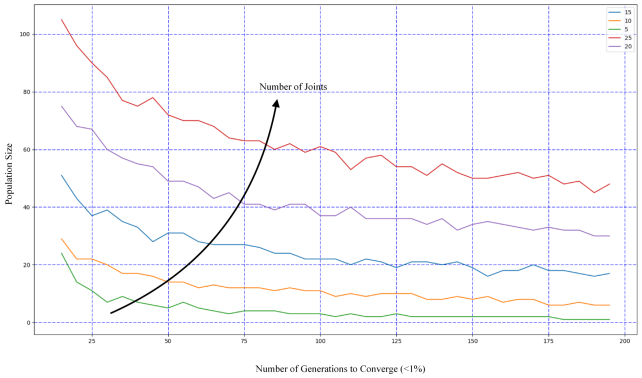

Figure 3: ***Population Size vs Generations*** *- Average over 100 iterations - Population Size vs Number of Generations required to converge on a solution (< 1%) for different numbers of serial linked chains. Single fitness criteria (distance of the end-effector from the target location).*

An important 'optimisation' factor for articulated character structures is the ability to tune 'aesthetic' qualities. For instance, coherent changes between poses, smooth overall variations (distributed averaging vs abruptly applying the solution to few links) even the ability to best guess or match reference key-frame examples. Figure 2 shows a simple proof of concept whereby the structure is smoothed at different ends of the chain (secondary priority) when finding the end-effector reach location.

The centre-of-mass is often an overlooked property in articulated kinematic problems, due to its challenging and costly overhead. As shown in Figure 4 and Figure 5 the importance of the physical properties of the character for generating 'coordinated' balanced poses.

Weighted Sum Fitness Functions Evolutionary algorithms operate on a population and are well suited to solving multiple solutions in parallel (i.e., they are readily able to adapt to multi-objective optimisation problems). The weighted sum of the different fitness functions is a fairly standard way to handle multiple objectives. Then again, the final answer does depend greatly on the set priorities. We manage the priorities using a weighted fitness factor. The weighted fitness function is defined in Equation 1 as:

$$fitness^{total} = \sum_{j=1}^{n} w_j f_j \tag{1}$$

where n is the number of fitness criteria, w_j denotes the relative weight of importance for the fitness criterion f_j. Examples of the different criteria include, end-effector orientation and location,

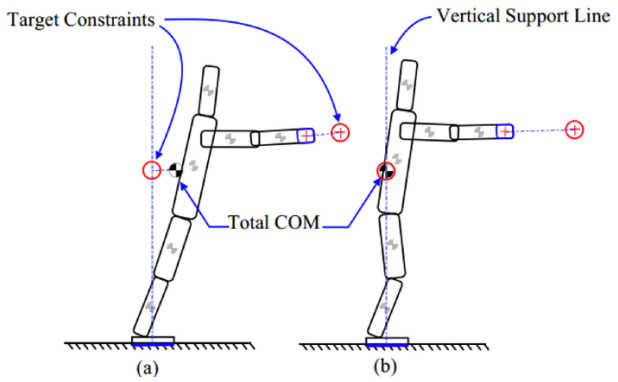

Figure 4: ***Centre-of-Mass (COM) Character Body*** *- The importance of the COM constraint (a) COM and end-effector constraint 'equal' priority and (b) Setting the COM as priority in the weighted sum objective function.*

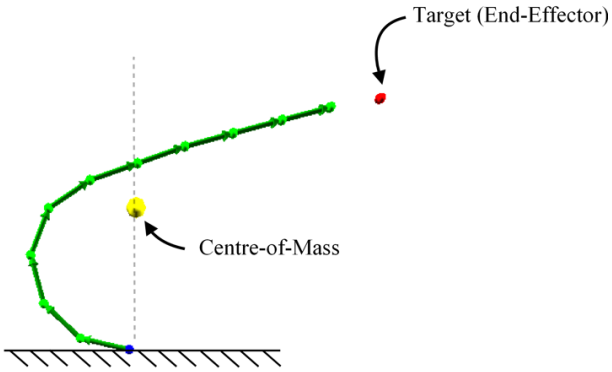

Figure 5: ***Centre-of-Mass Priority*** *- The centre-of-mass (COM) is crucial for dynamic systems. We set three fitness criteria: highest priority is the COM must remain above the support point, second is the end-effector reach target, and finally the pose should coherent between joints (i.e., avoid abrupt jumps).*

comfort factors, self-collision, obstacle avoidance and even overall centre-of-mass position (e.g., see Equation 2).

$$\begin{aligned}\text{Fitness} =& (w_0)(f_{ee}) \\ &+ (w_1)(f_{com}) + (w_2)(f_{style}) + \ldots\end{aligned} \tag{2}$$

where w_x represent the weights and are in the range [-1.0..0.0..1.0] and f_{ee}, f_{com}, and f_{style} are fitness values. The weights determine how important the fitness value is in achieving the final objective (trade-off objectives). As shown in the following fitness calculations, we normalize the fitness criteria using an exponential curve (i.e., $1 - exp(-||x||)$) to limit the fitness function between 0.0 and 1.0.

The overall centre-of-mass represents a unique point for an object (or system) which is used to describe the object's physical characteristics, such as, the object's ability to respond to external forces and torques (see Equation 3).

$$\begin{aligned}M &= \sum_{i=1}^{N} m_i \\ \vec{P}_{COM} &= \frac{\sum_{i=1}^{N} m_i \vec{P}_i}{M}\end{aligned} \tag{3}$$

where M is the total mass, m_i is the mass of joint i, N is the number of joints, $\vec{P}_{COM}$ is the position of the centre-of-mass, and $\vec{P}_i$ is the position of mass m_i.

The fitness calculation between the current and desired overall centre-of-mass location (Equation 4):

$$f_{com} = 1 - exp(-||\vec{P}_{COM_{cur}} - \vec{P}_{COM_{des}}||) \tag{4}$$

where f_{com} is the fitness value for the overall centre-of-mass (0.0 to 1.0), $\vec{P}_{COM_{cur}}$ is the current, $\vec{P}_{COM_{des}}$ desired positions of the overall centre-of-mass.

The end-effector error is the difference between the desired and current end-effector location (Equation 5):

$$f_{ee} = 1 - exp(-||\vec{P}_{cur} - \vec{P}_{des}||\mu) \tag{5}$$

where f_{ee} is the fitness value for the end-effector location, $\vec{P}_{cur}$ is the current end-effector location, $\vec{P}_{des}$ is the desired end-effector location. We subtract the negative exponent from 1.0 to limit the fitness function to 0.0 to 1.0, and μ is a scaling factor to deal with numerical issues due to the exp calculation (e.g., $\mu < 1.0$)

The comfort fitness calculation compares the current joint angles with a set of reference angles (see Equation 6).

$$\begin{aligned}\theta_{etotal} &= \sum_{i}^{N} ||\theta_{curr_i} - \theta_{des_i}|| \\ f_{sumstyle} &= 1 - exp(-\theta_{etotal}\mu)\end{aligned} \tag{6}$$

where f_{style} is the fitness value for the quality of the pose, N is the number of angles, θ_{curr} is the current joint angle, θ_{des} is the desired joint angle and μ is a scaling factor to deal with numerical issues due to the exp calculation (e.g., $\mu \approx 0.1$). For example, when re-targeting key-frame poses in animation data, we want the adapted pose to remain as close to the original reference pose as possible while meeting the specified criteria (e.g., feet on the ground or location of the centre-of-mass for balance). There are different methods for accomplishing the joint-different penalty, for example, Rodriguez et al. [RSRHO] stores only the single maximum rotational error for all the joint angles compared to our approach that sums the error (shown in Equation 7). We compare the two fitness criteria in Figure 6. Since only the largest error is stored and compared against, for small numerical movements there is no perceivable feedback into the population to steer the algorithm causing long delays between updates.

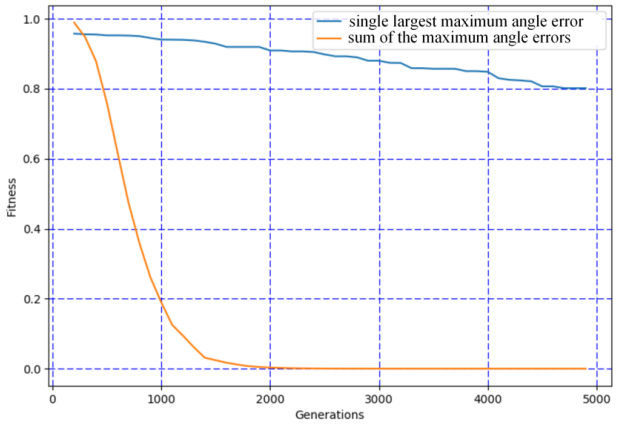

Figure 6: ***Comfort Fitness*** *- The DE algorithm performs better with gradually changing fitness functions compared to binary (step-wise) solutions.*

if *(abs(θ) > 2π)* **then**
  δ = abs(θ) - 2π
  $\theta = \theta \left(\frac{1.0f}{\delta} \right)$
end
Algorithm 2: Penalty feedback for constraining joint angle ranges ($\pm 2\pi$).

$$\begin{aligned} \theta_{err} = max(&||\theta_{curr0} - \theta_{des0}||, \\ &||\theta_{curr1} - \theta_{des1}||, \\ &||\theta_{curr2} - \theta_{des2}||, \\ &...) \\ f_{maxstyle} = 1 - exp(&-\theta_{err}) \end{aligned} \tag{7}$$

where θ_{curr0} is the current joint angle for joint 0, θ_{des0} is the desired joint angle for joint 0.

Joint angles are limited to $-\pi$ to π. As the algorithm evolves, the values will violate these limits. Instead of hard clamping the values - we introduce a penalty feedback.

Massively Parallel Architecture The sequential differential evolutionary algorithm in its original form is difficult to map to a massively parallel architecture such as the Graphical Processing Unit (GPU). Our modified algorithm and optimisation problem is highly suited to the GPU due to the fixed structure of the computation (i.e., the kinematic composition is the same for each individual in the population). A sequential implementation shown in Algorithm 1, randomly selects individuals from the population, hence, inside the Epoch update, an individual candidate may be overwritten multiple times in a single step. This would cause problems on the GPU since neighbouring individuals would be changing - and if this was to be implemented on the GPU would require data to be locked while different processes read/write to elements. However, we modify the algorithm so it is more compatible with the GPU architecture. We have 'two' blocks of memory for the population -

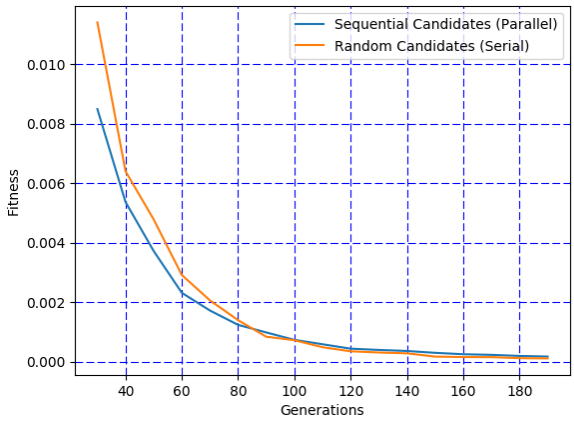

Figure 7: ***Random vs Sequential*** *- Removing the 'random' candidate selection component from the sequential version of the algorithm 1. Plot the average fitness (100 iterations) vs population for 20 joints (Population = 500).*

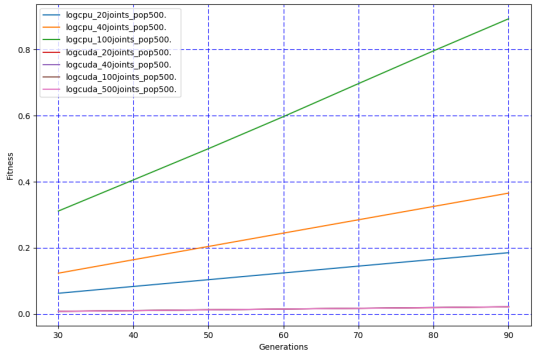

Figure 8: ***Timing Information*** *- Population of 500 for varying numbers of joints (i.e., degrees of freedom). Plot of the time vs number of generations for both the CPU and GPU implementation.*

the current population and the next population. Each Epoch update takes the current population as the source for 'reading' from and use it to calculate the 'next' population (i.e., the differential evolutionary technique is used to calculate each individual for the subsequent population). After the Epoch update, we 'swap' the memory so the next population becomes the current and the current population becomes the next. Essentially, 'ping-ponging' back and forth - rapidly evolving the population without any 'bottlenecks' to hinder the computations (see Figure 1).

In contrast with a similar evolutionary algorithm known as the genetic algorithm - the differential evolutionary algorithm has the added advantage of not requiring 'sorting' of the population after each Epoch to identify the elitist individuals - enabling us to quickly evolve the solution over multiple generations.

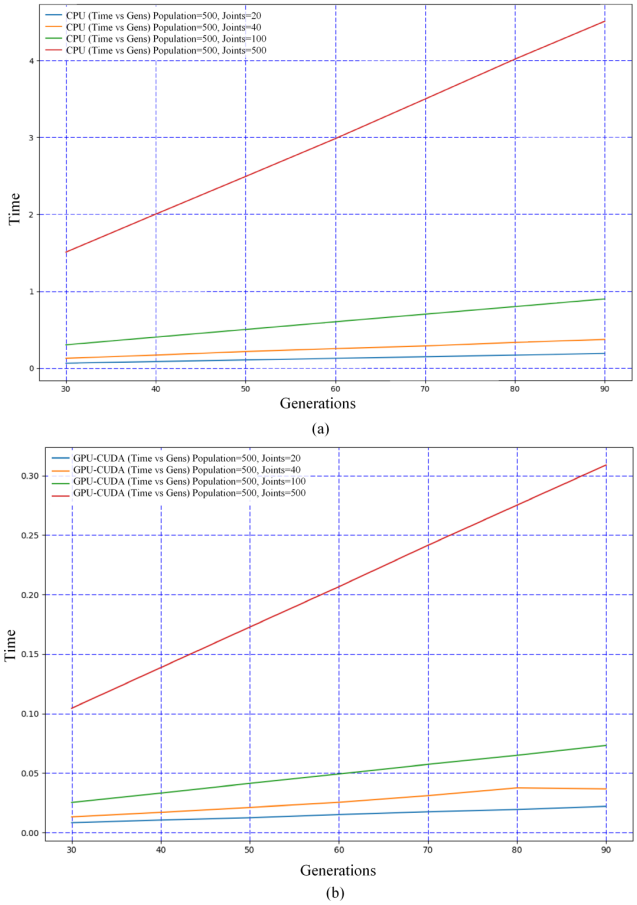

Figure 9: ***Separate Timing Data - Serial (CPU) vs Parallel (GPU)*** *- Comparison of the differential evolutionary algorithm timing information for (a) CPU and (b) GPU (averaged over 100 iterations). On average, with a population size of 500 and each individual having 100 links the parallel CUDA implementation was 12x faster than the serial (CPU) version.*

CUDA CUDA (Compute Unified Device Architecture) is an extension of the C programming language and was created by NVidia for programming massively parallel architectures [Nvi17]. Importantly, CUDA is well suited to massively parallel problems (e.g., articulated character inverse kinematic problems). The GPU is an attractive solution for problems that require the calculation of lots of operations with floating point variables in parallel (i.e., while not requiring too much access to global memory). The GPU's internal memory registers offer access to hundreds or thousands of computing cores. Since CUDA is based on the SIMD (Single Instruction Multiple Data) paradigm, it is well suited to our inverse kinematic problem which requires the execution of the same operation on different data. As we show in this paper, as long as the fitness criteria does not have a substantial number of conditional branches or have considerable dependencies between neighbouring threads the implemented parallel solution is able to outperform the sequential version.

4 Conclusion

This paper has shown the promising potential of using a differential evolutionary algorithm for solving multi-objective kinematic problems. What is more, combining the technique with a massively parallel architecture helps reduce search times. For example, with a population size of 500 and each individual having 100 links the parallel CUDA implementation was 12x faster than the serial (CPU) version (see Figure 9). While we only applied the DE algorithm to small selection of character kinematic problems, the solution has a wide range of potential applications - especially when combined with a massively parallel execution environment - allowing the technique to converge on solutions in viable time frames. In addition, we explored various test scenarios, but later work will explore both static and dynamic situations (e.g., optimising multiple articulated poses to predict future motions).

Possible future directions of the research presented in this paper include: Firstly, multiple sub-populations to ensure the massively parallel implementation (on the GPU) is kept continually working at 100% (i.e., no waiting or stalls between population updates) which leads onto possibly extending this to multiple GPUs and clusters. Secondly, we focused on adapting a single articulated structure (i.e., single pose), however, the technique could be extended to process multiple-frames simultaneously (coherency and fitness criteria between frames - e.g., motion frequency analysis [Ken15]). Finally, depending upon the complexity of the model, it might be possible to extend the optimisation criteria beyond a purely kinematics problem to include physically-based simulation data (i.e., for low-dimensional approximations).

Acknowledgements

We gratefully acknowledge the support of the NVIDIA Corporation for the donation of hardware used in this research. In addition, we want to thank the reviewers for taking the time to provide insightful and valuable comments to help to improve the quality of this paper.

References

[AKT*08] ALBERT F., KOH S., TIONG S., CHEN C., YAP F.: Inverse kinematic solution in handling 3r manipulator via real-time genetic algorithm. In *Information Technology, 2008. ITSim 2008. International Symposium on* (2008), vol. 3, IEEE, pp. 1–6. 2

[BB04] BAERLOCHER P., BOULIC R.: An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The visual computer 20*, 6 (2004), 402–417. 2

[BK05] BUSS S. R., KIM J.-S.: Selectively damped least squares for inverse kinematics. *journal of graphics, gpu, and game tools 10*, 3 (2005), 37–49. 2

[Ble17] BLENDER: Blender. *URL: www.blender.org (Accessed: (05/02/2017))* (2017). 1

[Deb00] DEB K.: An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering 186*, 2 (2000), 311–338. 2

[DVS01] D'SOUZA A., VIJAYAKUMAR S., SCHAAL S.: Learning inverse kinematics. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on* (2001), vol. 1, IEEE, pp. 298–303. 2

[DW97] DEO A. S., WALKER I. D.: Minimum effort inverse kinematics for redundant manipulators. *IEEE Transactions on Robotics and Automation 13*, 5 (1997), 767–775. 2

[GBM12] GONZÁLEZ C., BLANCO D., MORENO L.: A memetic approach to the inverse kinematics problem. In *Mechatronics and Automation (ICMA), 2012 International Conference on* (2012), IEEE, pp. 180–185. 3

[GC00] GEN M., CHENG R.: *Genetic algorithms and engineering optimization*, vol. 7. John Wiley & Sons, 2000. 2

[GMHP04] GROCHOW K., MARTIN S. L., HERTZMANN A., POPOVIĆ Z.: Style-based inverse kinematics. In *ACM transactions on graphics (TOG)* (2004), vol. 23, ACM, pp. 522–531. 1

[KAAT03] KALLMANN M., AUBEL A., ABACI T., THALMANN D.: Planning collision-free reaching motions for interactive object manipulation and grasping. In *Computer Graphics Forum* (2003), vol. 22, Wiley Online Library, pp. 313–322. 2

[Kal08] KALLMANN M.: Analytical inverse kinematics with body posture control. *Computer animation and virtual worlds 19*, 2 (2008), 79–91. 2

[Ken12a] KENWRIGHT B.: Inverse kinematics–cyclic coordinate descent (ccd). *Journal of Graphics Tools 16*, 4 (2012), 177–217. 2

[Ken12b] KENWRIGHT B.: Synthesizing balancing character motions. In *VRIPHYS* (2012), pp. 87–96. 1

[Ken14] KENWRIGHT B.: Planar character animation using genetic algorithms and gpu parallel computing. *Entertainment Computing 5*, 4 (2014), 285–294. 2

[Ken15] KENWRIGHT B.: Quaternion fourier transform for character motions. 7

[KSB10] KUMAR S., SUKAVANAM N., BALASUBRAMANIAN R.: An optimization approach to solve the inverse kinematics of redundant manipulator. *International Journal of Information and System Sciences (Institute for Scientific Computing and Information) 6*, 4 (2010), 414–423. 2

[Nvi17] NVIDIA C.: Compute unified device architecture programming guide. 7

[OLMY14] OMIDVAR M. N., LI X., MEI Y., YAO X.: Cooperative coevolution with differential grouping for large scale optimization. *IEEE Transactions on Evolutionary Computation 18*, 3 (2014), 378–393. 2

[PKG89] PARKER J. K., KHOOGAR A. R., GOLDBERG D. E.: Inverse kinematics of redundant robots using genetic algorithms. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on* (1989), IEEE, pp. 271–276. 2

[Pri96] PRICE K. V.: Differential evolution: a fast and simple numerical optimizer. In *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American* (1996), IEEE, pp. 524–527. 2

[QHS09] QIN A. K., HUANG V. L., SUGANTHAN P. N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation 13*, 2 (2009), 398–417. 2

[RSRHO] RODRIGUEZ E., SAHA B. N., ROMERO-HDZ J., ORTEGA D.: A multi-objective differential evolution algorithm for robot inverse kinematics. *SSRG International Journal of Computer Science and Engineering (SSRG - IJCSE)*. 3, 5

[SAC14] SON N. N., ANH H. P. H., CHAU T. D.: Inverse kinematics solution for robot manipulator based on adaptive mimo neural network model optimized by hybrid differential evolution algorithm. In *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on* (2014), IEEE, pp. 2019–2024. 3

[SJTJ15] SUI Z., JIANG L., TIAN Y.-T., JIANG W.: Genetic algorithm for solving the inverse kinematics problem for general 6r robots. In *Proceedings of the 2015 Chinese Intelligent Automation Conference* (2015), Springer, pp. 151–161. 2

[SP97] STORN R., PRICE K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization 11*, 4 (1997), 341–359. 2

[TGB00] TOLANI D., GOSWAMI A., BADLER N. I.: Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical models 62*, 5 (2000), 353–388. 2

[VT04] VESTERSTROM J., THOMSEN R.: A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC2004. Congress on* (2004), vol. 2, IEEE, pp. 1980–1987. 2, 3

[Yan12] YANG X.-S.: Chaos-enhanced firefly algorithm with automatic parameter tuning. *Int J Swarm Intell Res 2*, 4 (2012), 125–36. 2