ARTICLE IN PRESS

Computer Physics Communications ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

A parallel algorithm for random searches

M.E. Wosniack [a,*], E.P. Raposo [b], G.M. Viswanathan [c], M.G.E. da Luz [a]

[a] Departamento de Física, Universidade Federal do Paraná, C.P. 19044, 81531-980 Curitiba-PR, Brazil

[b] Laboratório de Física Teórica e Computacional, Departamento de Física, Universidade Federal de Pernambuco, 50670-901 Recife-PE, Brazil

[c] Department of Physics and National Institute of Science and Technology of Complex Systems, Universidade Federal do Rio Grande do Norte, 59078-900 Natal-RN, Brazil

ARTICLE INFO

Article history:
Received 6 May 2015
Received in revised form
24 July 2015
Accepted 25 July 2015
Available online xxxx

Keywords:
Random search
Parallel random search
Parallel random walk
Lévy flights

ABSTRACT

We discuss a parallelization procedure for a two-dimensional random search of a single individual, a typical sequential process. To assure the same features of the sequential random search in the parallel version, we analyze the former spatial patterns of the encountered targets for different search strategies and densities of homogeneously distributed targets. We identify a lognormal tendency for the distribution of distances between consecutively detected targets. Then, by assigning the distinct mean and standard deviation of this distribution for each corresponding configuration in the parallel simulations (constituted by parallel random walkers), we are able to recover important statistical properties, e.g., the target detection efficiency, of the original problem. The proposed parallel approach presents a speedup of nearly one order of magnitude compared with the sequential implementation. This algorithm can be easily adapted to different instances, as searches in three dimensions. Its possible range of applicability covers problems in areas as diverse as automated computer searchers in high-capacity databases and animal foraging.

1. Introduction

Surely, computational simulations constitute a keystone tool for our scientific understanding of nature [1]. However, given the diversity and the potential complexity [2] of necessary (numerical) models to study distinct phenomena, the optimization of the underlying algorithms becomes, in many instances, a crucial aspect [3] in their viability.

Towards this optimization goal, parallel computing strategies [4] are particularly important. Usually, algorithms and codes that rely on sequential decisions are not easily parallelizable, like P-complete problems [5,6]. But in spite of that, some processes thought to be inherently sequential, like the depth-first search [7], have been solved in a parallel fashion thanks to a proper distribution of work between the processors [8]. Other highly sequential instances, as edge coloring [9] and the maximal independent set [10], also have found alternative parallel solutions. Further, parallelized algorithms may be constructed in a very different way from the sequential counterparts, like parallel genetic algorithms that have a super-linear performance when compared to their sequential versions [11]. A collection of multidisciplinary problems allowing parallel approaches can be found in [12].

The general random search problem consists of finding a competent strategy for the encounter of randomly located target sites that can only be detected in the limited vicinity of the searcher. Its possible range of applicability covers areas as diverse as automated computer searchers of registers in high-capacity databases [13], motion of binding enzymes or proteins along DNA [14], economics [15], operational research like hunt for submarines [16], and animal foraging [17,18]. In certain contexts, such as in animal foraging [18], the knowledge of the distribution of encountered targets provides an important way to characterize how the resources are exploited during the search (e.g., if in an efficient manner).

In dealing with random search through numerical models [17,18], one often faces difficulties concerning the long time it takes to obtain meaningful results, a consequence of the large number of averages required. Actually, in this area of research [17,18] the computational procedures are traditionally sequential: the random walker moves from target to target until some halt criterion is achieved. The question is then how to formulate exactly the same problem, nevertheless using a framework allowing parallelization (with a consequent reduction of computational resources and time). One possibility is instead to consider a single

* Corresponding author.
E-mail address: wosniack@fisica.ufpr.br (M.E. Wosniack).

http://dx.doi.org/10.1016/j.cpc.2015.07.014
0010-4655/

random walker looking for Q targets, to assume Q random walkers looking for only one target. But for both models to lead to equal results, a carefully constructed extra condition must be imposed to the latter version. This extra information – in the form of initial conditions – concerns the spatial pattern that arises from the targets detected by a sole searcher, like footprints left by the walker on the original distribution of targets during the random search. Our present parallelized solution to the problem is based precisely on this idea. In fact, in order to recover the same distribution of detected targets observed in the sequential search routine, hence yielding the same statistical properties, our parallel implementation deals with a particular choice for the initial coordinates of each independent parallel random walker. This set of initial coordinates is actually built (as detailed below) from the original sequential search problem distribution of distances between two consecutively detected targets. It is basically a lognormal curve, whose mean and standard deviation values fully depend on the search strategy and environment density considered.

As for the protocol technical implementation, the parallelization is accomplished using OpenMP directives in the original (sequential) C code [19]. In the serial code, we have identified which functions could be independently processed, and then have used parallel directives to split the work between the threads. Each thread is able to execute a set of instructions independently, namely, to access the environment (stored in the shared memory), and to proceed with its own random walk according to its private variables. The OpenMP approach has been chosen because its directives have a simple implementation and also provide a natural solution for the memory consistency problem that appears in the parallelization.

The paper is organized as follows. In Section 2 we give a very brief overview about parallelization of problems involving random walks. The features of typical sequential random searches (important for our goals) are described in Section 3. In Section 4 we detail the proposed parallel search algorithm and compare with results from the usual sequential simulation. Finally, few remarks and the conclusion are presented in Section 5.

2. Few examples of parallelization of random walks based algorithms

As we are going to show in the next sections, our parallel algorithm for the random search problem actually promotes a considerable speedup of the simulations runs. So, certainly it is a contribution of practical importance in the field. However, conceptually the method is likewise relevant. By including a dynamical constraint to a reformulation of the original process, we are able to make it amenable to a parallel construction. Therefore, we are adding a new example to the previously mentioned (and not so large) list of systems which are essentially sequential in character, but even then can be parallelized.

Before going into our specific problem, few comments about the parallelization of random walks in a broader perspective are in order next. Random walks have been important tools in Monte Carlo simulations. For instance, to achieve flat histograms to calculate the density of states, the Wang–Landau method employs independent random walkers for different energies [20,21]. It allows a proper numerical solution for larger systems, with the independent random walkers playing a crucial role in this respect. One possible parallelization of the protocol assigns one random walk for each available thread [22], using thousands of threads from the GPU. Nevertheless, the sampling over the energy landscape is not sufficiently homogeneous, thus being necessary to specify more random walkers to the lower energy regions. An OpenMP implementation [23] has been designed for the Ising model using the Wang–Landau method. Moreover, the parallelism can be explored in the solution of partial differential equations that use the Monte Carlo method [24] with multiple random walkers.

In graph theory, an important application of parallel random walks is the study of the cover time of a graph (i.e. the necessary time to visit all the nodes). It is known that for some graphs of size n the use of k random walkers (with $k \leq \log n$) can decrease the cover time by a factor of k [25]. However, the choice of the initial coordinates for the walkers influences both the cover and the hitting times of random regular graphs [26,27]. In this sense, there is an optimal choice of initial coordinates, which is dependent on the topology of the graph, and that minimizes both times. The $s-t$ connectivity problem, in which one has to determine if the vertices s and t are connected to a same component, also gains efficiency when several random walkers are initialized. In wireless networks, the use of multiple random walkers in search for a target reduces both the computing time and the network overhead [28].

Further, in model-checking algorithms, where one has to verify the correctness of a system implementation, parallel random walkers have been successfully employed to explore the states space looking for errors [29]. These independent random walks can explore more states in a fraction of the sequential time, since the revisits to the same location are decreased in such arrangement [30]. Some model-checking algorithms are designed with coordinate random walkers, that simulate properties of animal foraging. The BEE algorithm [31], inspired by the cooperative behavior in bee hives, allocates parallel random walkers in regions of errors that were previously identified and communicated by an exploring random walker. The process is similar to the scouting of idle working bees when one bee identifies a profitable flower patch and informs its location to the colony. Another model with biological inspiration uses ant robots that work in a parallel and decentralized fashion, interacting only locally to explore a landscape efficiently [32].

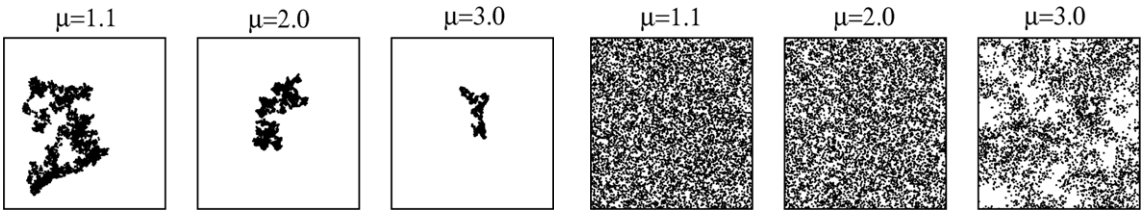
All these examples illustrate the great computational gain in developing parallel procedures for algorithms based on random walks and searches. The present contribution goes exactly in this direction.

3. Sequential search properties

In order to build a parallel version of the random search problem, we present in this section some properties of the sequential version of the code that are necessary for our parallel implementation. The sequential random search model is discussed in detail in Ref. [18].

The search space is a square region of size D with periodic boundary conditions. In this environment a given amount of targets is distributed in a homogeneous way, with average distance l_t between them. The value of l_t (measured in terms of the radius of vision r_v, see below) characterizes the density of targets: the larger (smaller) l_t, the lower (higher) the density. The targets are non-destructive, i.e. they can be detected an unlimited number of times during the search. The searcher is a random walker whose step lengths ℓ are taken independently from a probability density distribution $P(\ell)$ at a random direction. The interaction between the searcher and the targets is provided by the radius of vision r_v that defines the region around the searcher where the target can be detected (here we set $r_v = 1$). Along the step j of length ℓ_j the walker constantly looks for targets within a distance r_v. If no target is found, the searcher completes the full step. Otherwise, a target is detected and the step ℓ_j is truncated. This process is then repeated with a new step and direction taken until the stop (halt) condition for the simulation is reached. For the probability density function of the step length ℓ we consider a power-law $P(\ell) \sim \ell^{-\mu}$ ($1 < \mu \leq 3$) for $r_v < \ell < D$ and 0 otherwise. This power-law distribution corresponds to the long-distance limit of the family of α-stable Lévy distributions with index $\alpha = \mu - 1$ [18]. The

Please cite this article in press as: M.E. Wosniack, et al., A parallel algorithm for random searches, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.07.014



(a) High density: $l_t = 10$. (b) Low density: $l_t = 100$.

Fig. 1. Illustration of the spatial patterns of the positions of $T = 10^4$ detected targets under ballistic ($\mu = 1.1$), superdiffusive ($\mu = 2.0$) and Brownian ($\mu = 3.0$) random search strategies. (a) Dense and (b) sparse environments, with l_t denoting the average distance between targets. In the high-density regime, the presence of larger steps in the ballistic walk allows to detect targets distributed over a larger area. In contrast, a Brownian strategy tends to confine the searcher to a narrower region in this regime. On the other hand, when the targets density is low the large number of encounters ($T = 10^4$) chosen as the stop criterion makes the distribution of targets found to spread nearly homogeneously over the whole search space for $\mu = 1.1$ and $\mu = 2.0$, whereas some empty regions can be observed in the Brownian searches due to the small displacements.

lower limit of the step lengths prevents the searcher to waste a step in a region that it already knows (using its visual radius) and that does not have targets. On the other hand, the upper limit is a consequence of the periodic boundary conditions: if a step longer than D (throughout the work we set $D = 10^4$) is allowed, the searcher can in principle explore the same space more than once in the same step. But very important, although the steps lengths here are truncated, to a great extent they behave as usual non-truncated Lévy processes for very long times [33]. So, the exponent μ provides different search behaviors, from a ballistic superdiffusive walk ($\mu \to 1$) to the normal diffusion Brownian motion ($\mu > 3$). One of the main results concerning power-law random searches in non-destructive sparse distributions is that the walker with strategy $\mu \approx 2$ maximizes the search efficiency, a result with empirical and analytical support [17,18,34].

A particular aspect of random searches that has not attracted much attention in the literature is the spatial pattern formed by the detected targets. An analytical approach has been developed in Ref. [35], but considering several random walkers with a common starting position, instead of the resulting pattern of one random walker. From a practical perspective, in the foraging activity of distinct animal species each one explores the resources in a different spatial scale: some of them have a dispersal rate higher than the others [34]. In our model, each search strategy μ leaves a particular footprint on the environment. For example, ballistic searches explore a larger region of the space if compared to Brownian searchers, as a consequence of the distribution of step lengths. Moreover, when the density is varied, the spatial pattern of the found targets also changes accordingly. These properties are summarized in Fig. 1, in which a total of $T = 10^4$ encounters are registered in each search.

To characterize the particular spatial pattern associated with each search strategy, we first focus on the behavior of the distribution of distances between two sequentially detected targets. We record such distances as the straight lines that connect the targets. Null distances corresponding to the return to the target just visited are not included in the statistical analysis. Since the environment has periodic boundary conditions, for each couple of distinct points successively visited two possible values for the distance between them arise. For convenience, we always consider the shortest distance in our records. We actually work with the logarithm of the distances because this quantity seems to follow a normal distribution under certain conditions, as can be seen in Fig. 2. We also keep the information about the number of revisited targets for a later analysis.

The set of histograms displayed in Fig. 2 corresponds to the same parameters considered in the patterns of Fig. 1. Except for Fig. 2(a), the data seem to closely follow a normal-shaped distribution with definite parameters mean m and standard deviation σ. Even though the fitting is not accurate, working with an approximate distribution is sufficient in our method. Since superdiffusive search strategies have a steps distribution with both short and long components that depends on its search exponent μ, the resulting interaction between the walker and the targets is not uniform. The searcher will accomplish detections both close to its last visited target and far away from it. However, when $\mu = 1.1$, the frequency of long steps is very high, and in a low density scenario (say, $l_t = 100$) the detected targets are usually very distant from each other. The histogram in the inset of Fig. 2(a) shows the (original) distribution of distances prior to the logarithm transformation. It does not display the typical heavy tail of lognormal distributions, presenting instead a low decay (specially for distances up to $D/2 = 5\times10^3$, being almost uniform in such region). This contrast with Fig. 2(b)–(f), where both distant and close targets are found (but at different proportions) and the lognormal behavior is observed. When the targets density increases, even the ballistic searcher will tend to detect targets in the neighborhood of its last detection, and the distances distribution shows a lognormal shape. But we should emphasize that to represent this a bit harder $\mu = 1.1$, $l_t = 100$ case (given rise to a homogeneous distribution of detected targets, see Fig. 1(b)) by a normal-shaped distribution with a large mean – thus consistent with Fig. 3 – already turns out to be good enough for our later purposes.

From simulations using the sequential code, we obtain the parameters that characterize the normal-shaped distribution showed in Fig. 2 for different targets density and search strategies. Working with the logarithm of the distances, we take advantage on the additive properties of the normal distribution and thus can perform averages over different spatial configurations. Indeed, each sequential search is averaged over $N = 2.5\times10^3$ simulations, with each run ending upon the finding of 10^4 targets. In Fig. 3 we display the behavior of the mean m and the standard deviation σ of the logarithm of the distances between targets consecutively found as a function of the exponent μ and for different configurations of the search space. As expected, Fig. 3(a) shows that the targets detected by the ballistic searcher are generally much further away than those encountered by the Brownian walker. Also, as it should be, the values for the mean m increase when sparser configurations are considered.

The standard deviation has a more complex behavior. We first observe in Fig. 3(b) that it reaches higher values for denser configurations, indicating that larger fluctuations around the mean are present for smaller average distances between targets. We should point out that even though the average distance between two targets is smaller in the dense regime, sometimes during the long search for $T = 10^4$ targets the searcher travels a large path in order to make a detection, especially for lower values of μ.

Another peculiarity in Fig. 3(b) is the curve for $l_t = 100$: it is maximized around $\mu = 2.0$, in contrast with the curves for the other (smaller) values of l_t. This effect is a consequence of an important feature of the $\mu = 2.0$ strategy in a sparse (large l_t) environment: it does not only detect both distant and nearby targets,

Please cite this article in press as: M.E. Wosniack, et al., A parallel algorithm for random searches, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.07.014

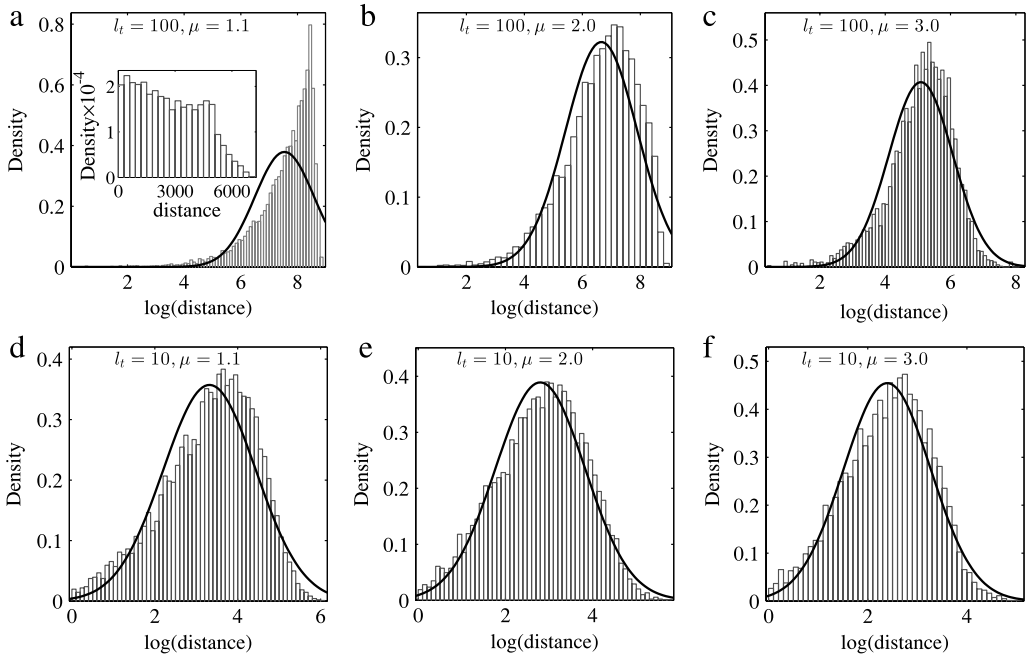


Fig. 2. Histograms of the logarithm of the distances between successively detected targets, corresponding to a single random search for $T = 10^4$ targets, for different l_t's and search strategies μ's (the parameters here are the same as those in Fig. 1). The original sets are reasonably well fitted by lognormal distributions (seen as Gaussians, continuous curves, because the logarithm rescaling), except in a scarce environment for a ballistic searcher, case (a). In the inset of (a) we have a histogram of the original distances, showing that they are fairly well distributed. Based on these analyses, one can "artificially" generate the same patterns of Fig. 1 by constructing a random walk with steps following the histograms (a)–(f).

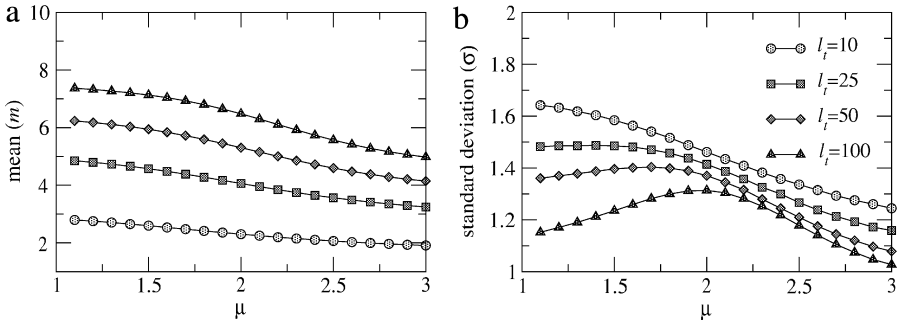


Fig. 3. (a) Mean m and (b) standard deviation σ of the logarithm of the distances between subsequently detected targets as a function of the search strategy μ. Each curve is the result of an average over $N = 2.5 \times 10^3$ walks, each one with $T = 10^4$ targets found. The density of the search space is indicated by the average distance between targets, namely $l_t = 10$ (circles), $l_t = 25$ (squares), $l_t = 50$ (diamonds), and $l_t = 100$ (triangles). The parameters m and σ are subsequently used to characterize the virtual environment for the parallel random searches (see text).

a signature of its super-diffusive behavior [36], but also it does so in a balanced way. Hence, the many scales are well represented, resulting in a larger standard deviation for this particular configuration. On the other hand, the strategy that is able to minimize both the mean and the standard deviation is $\mu = 3.0$ (due to its normal diffusion properties), always leading to detection of targets in the initial neighborhood of the searcher.

For our purposes, a final quantity also important to characterize the spatial distribution of detected targets is the fraction of those revisited during the search. We consider as a revisitation the instance when the searcher leaves the current target and then finds it again at some subsequent step, without the detection of distinct targets in between. The revisit events (whose parallel implementation is simple, see next section) take care of situations which are not included into the histograms in Fig. 2. In Fig. 4 we show the percentage of revisits during the search for $T = 10^4$ targets as a function of the average distance between targets l_t and the search strategy μ. The highest number of revisits occurs for $\mu = 3.0$, as expected for the normal diffusion regime. The density of targets also plays a role on this behavior: the sparser the environment (i.e., the larger the l_t) the higher the probability to return to the previously visited target. As it is going to be clear next, to know the number of revisitations in each random search configuration is fundamental to implement the parallelization procedure.

4. Implementing the parallelization

Now we discuss the parallel implementation of the random search problem and how the properties of the usual sequential case, Section 3, can be properly imposed in its construction.

4.1. The initial conditions for the parallel version

As already mentioned, the spatial pattern of detected targets (Fig. 1) is an important feature because it provides the information about how a single walker looking for a total of T targets progressively interacts with the environment. In the parallel version, we assume many independent random walkers (N_{rw}), each looking for Q targets (with $Q \ll T$), so that $N_{rw} \times Q = T$. To mimic the successive neighborhoods that a unique walker "sees" along the

Please cite this article in press as: M.E. Wosniack, et al., A parallel algorithm for random searches, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.07.014

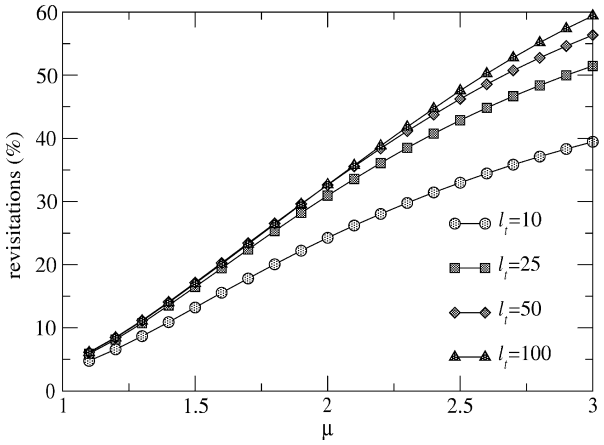


Fig. 4. Percentage of targets that are revisited during the random search to encounter $T = 10^4$ targets as a function of the search strategy (μ), in environments with distinct average distances between targets (l_t). Revisits to the last visited target are more frequent using a Brownian strategy ($\mu = 3$) in a sparse configuration ($l_t = 100$). Indeed, as a consequence of the normal diffusion, the Brownian searcher has a higher ($\approx 60\%$) probability to return to its starting position for $l_t = 100$. On the other hand, ballistic strategies ($\mu \to 1$) tend to access faraway regions of the search space, leading to a small number of revisits.

search, we use the parameters from Fig. 3 to generate the initial coordinates (i.e., *the starting positions* $\{(x, y)_{sp}\} = \{(x_1, y_1)_{sp}, \ldots, (x_{N_{rw}}, y_{N_{rw}})_{sp}\}$) of the N_{rw} parallel random walkers. In the way we define $\{(x, y)_{sp}\}$ (see below), any parallel walker n, starting from a point $(x_n, y_n)_{sp}$, while finding just few targets effectively will experience a stretch of the full trajectory traveled by a sole walker. Thus, collectively they properly sample the long term characteristic of the sequential search.

To construct a correct set $\{(x, y)_{sp}\}$ for each strategy and environment parameters μ and l_t, we first need to generate a lognormal distribution of corresponding m and σ, Fig. 3. Thus, we follow the standard protocol [37] of initially creating (with a random number generator) a list of normal distributed numbers $\{u\} = \{u_1, u_2, \ldots, u_{N_{rw}}\}$ with mean 0 and standard deviation 1. Next, we obtain another list of also normally distributed numbers $\{v\}$, with mean m and standard deviation σ, simply writing $v = u\sigma + m$. Then, we apply the mapping $z = \exp[v]$, getting a lognormal distribution $\{z\}$ similar to those displayed in Fig. 2. Second, for a given m and σ, we simulate an usual simple random walk (i.e., with no search), for which the turning angles are drawn from an uniform distribution and the jth step length is given by the element z_j of $\{z\}$. This walker takes in total N_{rw} steps and the coordinate positions of the successive steps end points form the set $\{(x, y)\} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{N_{rw}}, y_{N_{rw}})\}$.

Third, and lastly, two extra transformations are performed to the above $\{(x, y)\}$:

(i) The number of revisited targets during the sequential search, which can be inferred from Fig. 4, determines how many starting positions of the parallel walkers should coincide (a necessary restriction if the parallel implementation is going to reproduce patterns similar to those in Fig. 1). Thus, if for a certain μ and l_t the revisitation fraction is P_{rev}, we take the last $N_{rw} \times P_{rev}$ distinct coordinates in $\{(x, y)\}$ and randomly set them equal to the first $N_{rw} \times (1 - P_{rev})$ coordinate values. We notice this straightforward procedure (seeking to obtain the final $\{(x, y)_{sp}\}$) can work well for Q not too big. However, there is no reason to make Q very large since then we may start losing the advantages of the parallelization. Observe that in the parallel construction the walker can revisit its starting position and also detect a target in another walker's initial location, this last event corresponding to the sequential walker revisiting a target after detecting new ones.

(ii) In the sequential case, immediately after collecting a target located at, say (x_t, y_t), the walker initial position to look for the next target is obviously (x_t, y_t). Therefore, we substitute each location $j \equiv (x_j, y_j)$ of the set $\{(x, y)\}$ resulting from (i) by the coordinates of the closest target to j. In doing so, we end up with lists $\{(x, y)\}$ of positions which have exactly the motifs shown in Fig. 1. Finally, we take these sets as ours $\{(x, y)_{sp}\}$'s.

Fig. 5 summarizes the above described scheme for the initial conditions attainment. The transformations (i) and (ii) are illustrated in Fig. 5(b) and (c). Furthermore, we can see that the pattern of detected targets using the sequential, Fig. 5(a), and parallel, Fig. 5(d), algorithms are indeed very similar.

Finally, we mention the localization of the targets nearest to the coordinates in the list $\{(x, y)\}$ – procedure (ii) above – also should be optimized, otherwise the computational time gain with the parallelization process would be less effective. With such an aim, we first divide the environment into M quadrants. Then, we determine to which quadrant q ($q = 1, 2, \ldots, M$) a given (x_j, y_j) belongs to. Lastly, we look for the closest target to this point (x_j, y_j) by inspecting only q and its first neighbor quadrants. This operation also has been parallelized to increase the efficiency of the code: each thread receives a pair (x_j, y_j) of starting coordinates, identifies to which quadrant q this pair pertains, searches in q and in the adjacent quadrants for the closest target, calculating the distances coordinate-targets, and then selects the target with the shortest distance. The task is completed by the proper change of (x_j, y_j) by the closest target location.

4.2. The parallelized algorithm and the comparison between the parallel and sequential results

Prior to briefly describe the parallelized parts of the algorithm, we shall remark that some functions of the original code have been kept sequential, such as the initialization of the variables, the creation of the environment and the construction of the list with starting positions before the translation (transformation (ii)). So, still there is more room for further improvements. However, our main goal here is to show that the apparently essential sequential process of a single agent random search can actually be parallelized by means of proper adaptations in the problem original formulation. A fully optimized parallel protocol is presently under development and will be the subject of a future contribution.

The first parallel procedure is the generation of the initial conditions, detailed in Section 4.1. As mentioned, independently any thread must determine the closest target in its vicinity. The second parallelized process is the routine that actually performs the searches of the N_{rw} individual walkers (each having to find Q targets). A thread receives one initial position of $\{(x, y)_{sp}\}$ and the *full list* of available targets in the environment and then proceeds with the search task (making the probabilistic choices concerning the step lengths and turning angles). In order to further enhance the simulations, one could be tempted to consider just a fraction of the targets, say, only those in the close neighborhood of the searcher's starting location. However, this would not lead to a correct search pattern since, for instance, searchers with $\mu \to 1$ tend to hit targets rather distant from the departure point (see, e.g., the discussion in Ref. [36] as well as the exploitation patterns in Fig. 1). In this way, always all the targets must be made available to all the walkers.

In the simulations, the total traveled distance L_t for the detection of any target t is saved. After many encounters, in a total of T, the statistical search efficiency η is calculated, where the averages are taken over the results of all the threads. The quantity η is defined as [17,18] $\eta = T/\sum_{t=1}^{t=T} L_t$, thus basically being equal to

Please cite this article in press as: M.E. Wosniack, et al., A parallel algorithm for random searches, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.07.014

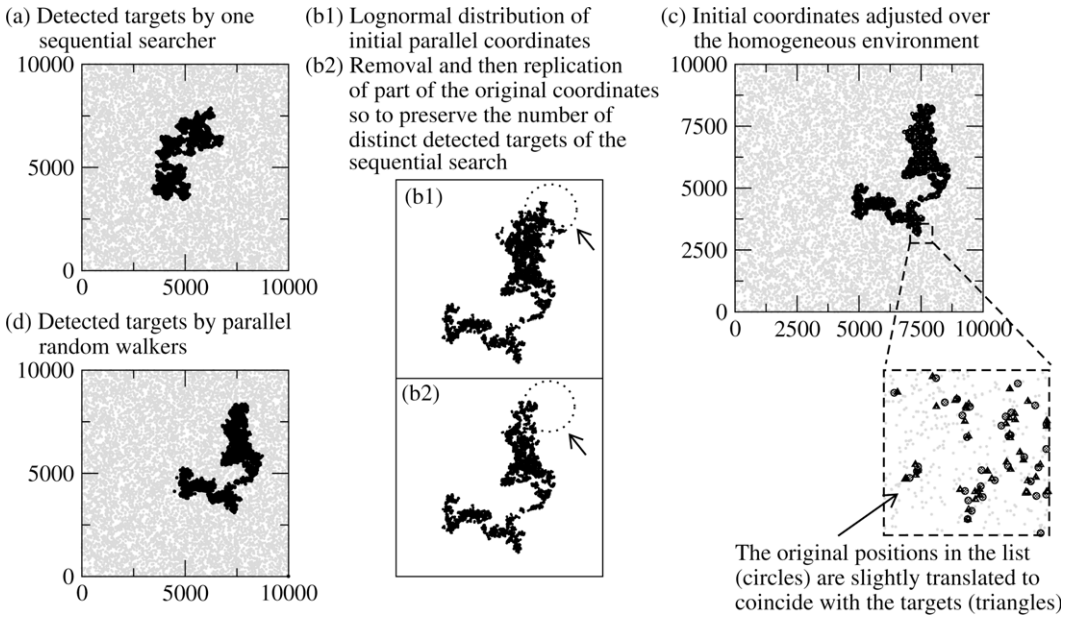


Fig. 5. Illustration of the initial conditions construction for the independent parallel walkers. The parameters here are $T = 10^4$, $\mu = 2.0$ and $l_t = 10$ (the two latters associated to lognormal mean and standard deviation $m = 2.71$ and $\sigma = 1.07$, see Fig. 3). (a) The complete environment and the detected targets (black dots) by a sequential searcher. (b1) As explained in the main text, the lognormal distribution of distances is used to generate the shown set of positions $\{(x, y)\}$. (b2) The set in (b1) goes through the procedure (i): the last $N_{rw} \times P_{rev}$ positions coordinates (indicated by arrows) are randomly substituted by the first $N_{rw} \times (1 - P_{rev})$ coordinate values. Hence, the number of distinct points displayed in (b2) matches the number of distinct detected targets in the pattern in (a). (c) The transformation (ii): the set of positions in (b2) are translated so as to coincide with the locations of the corresponding closest targets. (d) The found targets by all the random parallel searchers (which initiate their searches at the locations resulting from (c)). The patterns in (a) and (d) are very similar, e.g., notice their sizes, border shapes, and degree of compactness.

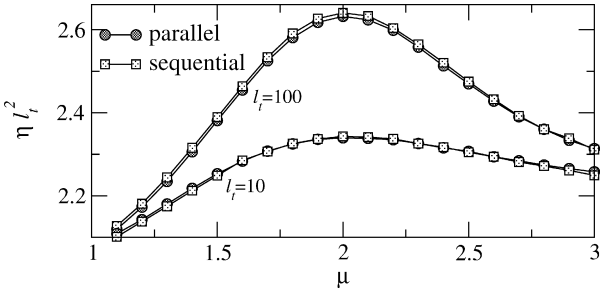


Fig. 6. Search efficiency (normalized by l_t^2) obtained from the parallel (circles) and sequential (triangles) implementations. It is clear that the sequential and parallel codes present practically the same numerical results. The stop condition is the detection of $T = 10^4$ targets in both dense ($l_t = 10$) and sparse ($l_t = 100$) environments. In the parallel code example here, each of the $N_{rw} = 10^4$ walkers looks for just a single target ($Q = 1$). The averages are performed over $N = 2.5 \times 10^3$ simulation runs.

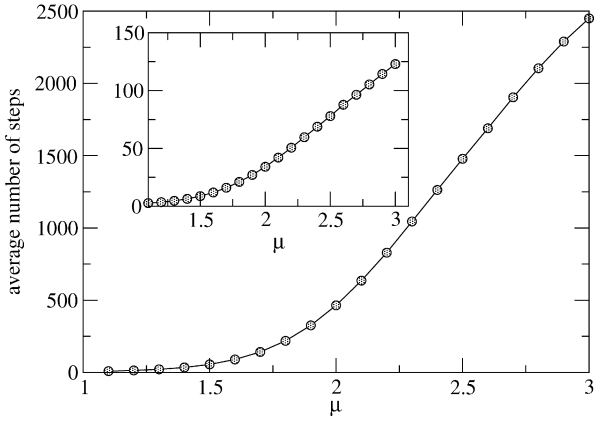


Fig. 7. For the sequential search protocol, the average number of steps necessary to encounter just one target as a function of the search strategy μ and for $l_t = 100$ and $l_t = 10$ (inset). In both low- and high-density regimes, the Brownian limit ($\mu \to 3$) demands a number of steps considerably larger to complete a same task, e.g., when compared to the ballistic strategy ($\mu \to 1$).

the inverse of the mean distance to find a target. In Fig. 6 we test the parallel implementation of the (single individual) random search problem comparing the function η obtained from the sequential and parallel codes. The efficiency curves are practically the same. Moreover, the optimal search strategy (maximum of η) perfectly coincides, arising for $\mu \sim 2$ as it should be the case [18]. So, from Fig. 6 (and also from Fig. 5(a) and Fig. 5(d)) we clearly see that the two codes yield the same statistical results. We shall mention that we have checked η from the parallel and sequential algorithms for a very large number of distinct parameters values (for instance, for Q ranging from 1 to 10 and so N_{rw} varying from 10^4 to 10^3), always obtaining the very good agreement observed in Fig. 6.

As a last remark, we point out that for μ around 3 the efficiencies here are slightly different from those in other previous publications (see, e.g., [17,36,38,39]). This has a simple cause. For convenience in comparing the results and also to make the parallel code easier to deal with in few technical aspects, for the environment construction (which is exactly the same in the parallel and sequential versions), we include an extra constrain in the targets creation: the distance between two targets is always larger than the visual radius r_v. As a consequence, the search space is not completely random (as commonly considered in the literature). There is a very short scale spatial correlation between nearby targets. Hence, the search efficiency for $\mu \to 3$ (just the case with more frequent short steps) might "feel" it, increasing a little the value of $\eta(\mu \sim 3)$ when contrasted to an environment without such restriction.

4.3. A simple benchmark analysis

In the sequential simulations, the total time spent to find T targets strongly increases with μ. This can be observed in Fig. 7, which displays the average number of steps necessary to complete the detection of a single target. Such a general trend takes places both at low and high target densities. Nevertheless, as expected

Please cite this article in press as: M.E. Wosniack, et al., A parallel algorithm for random searches, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.07.014

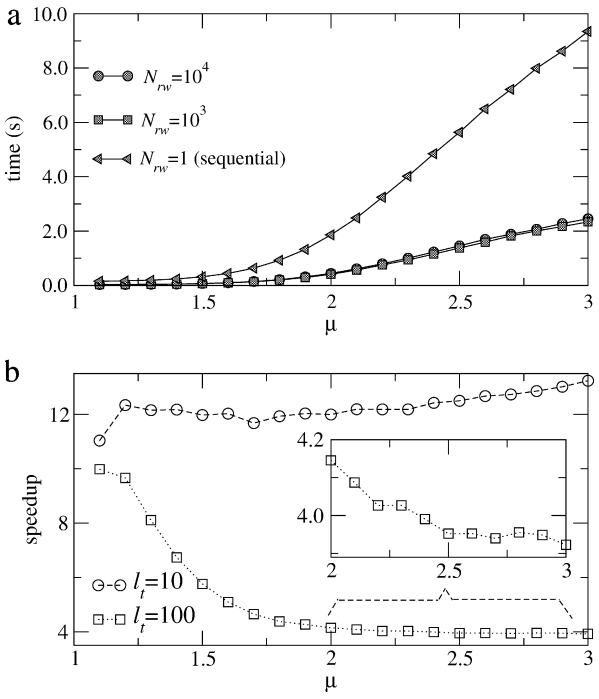


Fig. 8. Performance of the parallel code. (a) The elapsed simulation time (in seconds) required to find a total of $T = 10^4$ targets in a low density regime ($l_t = 100$). The task is performed by a single searcher $N_{rw} = 1$ (so $Q = 10^4$), i.e., a sequential random search, and for $N_{rw} = 10^3$ (with $Q = 10$) and $N_{rw} = 10^4$ (with $Q = 1$) independent searchers, hence the parallel implementation. Clearly, the sequential search is much more time consuming. In contrast, the difference in the elapsed time is not substantial for the two parallel examples presented. Simulations were performed in a CPU with 16 cores (32 threads). (b) The speedup ratio, namely, the ratio between the elapsed simulation times for the sequential and parallel runs, as function of the search strategy (μ), for dense ($l_t = 10$) and sparse ($l_t = 100$) regimes. For the high-density case, the resulting speedup is nearly constant and generally higher than that in the sparse landscape. For this latter, the speed up systematically decreases with μ (more strongly in the region $\mu < 2$ (see inset), thus in conformity with Fig. 7) due to the overhead generated when the threads need to perform a larger number of steps to encounter the targets, accessing the targets list more frequently.

the effect is more evident in the former: searchers with $\mu \to 3$ demand more than a thousand steps to detect a single target in a low density configuration, while those with $\mu \to 1$ need nearly a dozen steps to do so. For l_t decreasing by one order of magnitude (inset of Fig. 7) the difference between the $\mu \sim 1$ and $\mu \sim 3$ cases decreases by roughly also one order of magnitude.

Fig. 8(a) compares the sequential ($N_{rw} = 1$) and parallel simulation times. As the number of independent walkers, we consider $N_{rw} = 10^3$ and $N_{rw} = 10^4$. We recall that if one uses a relatively small number of parallel searchers (say $N_{rw} = 100$, so that $Q = 100$ if $T = 10^4$), the attempt to reproduce the sequential patterns in Fig. 1 through the collective behavior might be compromised. Moreover, the computational gain with the parallelization employing fewer N_{rw}'s tends to be less important. Finally, we note that the simulation time invariably increases with μ, in compliance with the results of Fig. 7.

Fig. 8(a) clearly illustrates the advantage of utilizing the parallel code. But to better quantify this, in Fig. 8(b) we show the parallelization speedup, defined as the ratio between the sequential and parallel elapsed times. We have that the speedup ratio is nearly independent on the search strategy (i.e., the value of μ) in dense environments, always above a tenfold gain. For sparser configurations, however, the computational speedup is around 10 only for ballistic searchers ($\mu \to 1$), decreasing to a value of about 4 for $\mu \gtrsim 2$. It can be understood in terms of the higher flux of accesses to the targets list in low-density environments (when it is more difficult to find a target). Indeed, in this case whereas superdiffusive walks ($\mu \leq 2$) can reach distant targets in just few steps, on the other hand each thread with $\mu > 2$ needs to read the target list (to check if there are targets along the way) much more often (because the steps are shorter, so in a larger number), leading to a lower speedup ratio.

5. Final remarks and conclusion

In this work we have presented a method to implement parallelization of the random search problem, a paradigmatic example of a sequential process. The procedure good performance has been verified by showing that it considerably reduces the computational time effort compared to the usual sequential simulations. Technically, the protocol relies on analyzing the pattern of detected targets in the sequential case and to ascribe to each parallel random walker a set of proper starting coordinates (based on such pattern). Then, these multiple searchers are allowed to perform the search independently from each other. The combined outcomes of all the parallel searchers reproduce very well the statistical properties of a single walker performing the full task. In fact, the numerical results obtained from the parallel implementation are exactly the same than those from a sequential simulation.

The most important new ingredient (compared to the usual sequential algorithm) is the construction of the initial conditions for the multiple independent searchers. This construction, despite the approximation in terms of a lognormal fitting, does yield very accurate numerical results. Furthermore, the use of the lognormal distribution is very useful because it leads to a great plasticity and a simple way to deal with distinct search landscape scenarios (being one of the reasons for the faster (and accurate) performance of the parallelized version of the random search). We should note that an alternative implementation of parallel random walks can be accomplished using Brownian bridges [40,41], that is, assigning initial and final coordinates for each random walker and generating a proper sequential trajectory. However, currently the method applies only to the Brownian case and our work addresses the superdiffusive context (in fact, the proper extension of Brownian to Lévy bridges would constitute an interesting topic of research). We also should mention that the parallel architecture chosen (OpenMP) has a pre-existent solution for the problem of several threads accessing the same data (besides being of easy usage).

The situations in which the computational gain is not so significant can be explained by the fact that often several threads try to access the same vector structure concurrently, causing thread overhead. But this problem could, in principle, be solved by more complex and involving algorithms (nevertheless, outside the scope of the present study).

Lastly, the procedure works nicely for homogeneous distributions of targets in two dimensions and the extension to the three dimensional case should be direct. Furthermore, conceivably the parallelization also could be accomplished when short range correlations [38] and limited memory effects in homogeneous target distribution [39] are present. On the other hand, if resources are fragmented into clusters, the identification of the patterns of detected targets becomes more difficult. Certainly, this would constitute the next challenge in trying to construct a general parallel algorithm. So, we hope that the present contribution can stimulate developments towards the design of parallel implementations that could be applied to random searches performed in arbitrary distribution of targets.

Acknowledgments

We thank CNPq, CT-Infra, CAPES, FACEPE and FAPERN for funding. Prof. Fabiano Silva is gratefully acknowledged for helpful discussion about parallelization techniques.

Please cite this article in press as: M.E. Wosniack, et al., A parallel algorithm for random searches, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.07.014

References

[1] A.B. Shiflet, G.W. Shiflet, Introduction to Computational Science: Modeling and Simulation for the Sciences, second ed., Princeton University Press, 2014.
[2] S. Arora, B. Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2009.
[3] C.H. Papadimitriou, Computational Complexity, John Wiley and Sons Ltd., 2003.
[4] R. Ciegis, D. Henty, B. Kågström, J. Žilinskas, Parallel Scientific Computing and Optimization: Advances and Applications, vol. 27, Springer Science & Business Media, 2008.
[5] R. Greenlaw, H.J. Hoover, W.L. Ruzzo, Limits to parallel computation: P-completeness theory, vol. 200, Oxford university press Oxford, 1995.
[6] J. Robson, Parallel Algorithms for NP-Complete Problems, in: Computer Science, Springer, 1992, pp. 379–382.
[7] J.H. Reif, Depth-first search is inherently sequential, Inform. Process. Lett. 20 (1985) 229–234.
[8] V.N. Rao, V. Kumar, Parallel depth first search. Part I, Implementation, Int. J. Parallel Program. 16 (1987) 479–499.
[9] H.J. Karloff, D.B. Shmoys, Efficient parallel algorithms for edge coloring problems, J. Algorithms 8 (1987) 39–52.
[10] R.M. Karp, A. Wigderson, A fast parallel algorithm for maximal independent set problem, J. ACM 32 (1985) 762–773.
[11] E. Alba, J.M. Troya, A survey of parallel distributed genetic algorithms, Complexity 4 (1999) 31–52.
[12] C. Bischof, Parallel Computing: Architectures, Algorithms, and Applications, vol. 15, IOS Press, 2008.
[13] P. Pirolli, S. Card, Information foraging in information access environments, in: G.C. van der Veer, C. Gale (Eds.), Proceedings of the 1995 Conference on Human Factors in Computing Systems, ACM, New York, 1995.
[14] P.H. von Hippel, O. Berg, Facilitated target location in biological systems, J. Biol. Chem. 264 (1989) 675–678.
[15] R.N. Mantegna, Lévy walks and enhanced diffusion in Milan stock exchange, Phys. A 179 (1991) 232–242.
[16] M.F. Shlesinger, Random searching, J. Phys. A 42 (2009) 434001.
[17] G.M. Viswanathan, S.V. Buldyrev, S. Havlin, M.G.E. da Luz, E.P. Raposo, H.E. Stanley, Optimizing the success of random searches, Nature 401 (1999) 911–914.
[18] G.M. Viswanathan, M.G.E. da Luz, E.P. Raposo, H.E. Stanley, The Physics of Foraging: An Introduction to Random Searches and Biological Encounters, Cambridge University Press, 2011.
[19] B. Chapman, G. Jost, R. van der Paas, Using OpenMP: Portable Shared Memory Parallel Programming, vol. 10, MIT Press, 2008.
[20] F. Wang, D.P. Landau, Efficient, multiple-range random walk algorithm to calculate the density of states, Phys. Rev. Lett. 86 (2001) 2050–2053.
[21] F. Wang, D.P. Landau, Determining the density of states for classical statistical models: A random walk algorithm to produce a flat histogram, Phys. Rev. E 64 (2001) 05601.
[22] J. Yin, D.P. Landau, Massively parallel Wang–Landau sampling on multiple GPUs, Comput. Phys. Comm. 183 (2012) 1568–1573.
[23] L. Zhan, A parallel implementation of the Wang-Landau algorithm, Comput. Phys. Comm. 179 (2008) 339–344.
[24] A. Youssef, A parallel algorithm for random walk construction with application to the Monte Carlo solution of partial differential equations, IEEE Trans. Parallel Distrib. Syst. 4 (1993) 355–360.
[25] N. Alon, C. Avin, M. Koucký, G. Kozma, Z. Lotker, M.R. Tuttle, Many random walks are faster than one, Combin. Probab. Comput. 20 (2011) 481–502.
[26] K. Efremenko, O. Reingold, How well do random walks parallelize? in: I. Dinur, K. Jansen, J. Naor, J. Rolim (Eds.), Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, in: Lecture Notes in Computer Science, vol. 5687, Springer Berlin Heidelberg, 2009, pp. 476–489.
[27] C. Cooper, A. Frieze, T. Radzik, Multiple random walks in random regular graphs, SIAM J. Discrete Math. 23 (2009) 1738–1761.
[28] B. Shah, K.-I. Kim, Towards enhanced searching architecture for unstructered peer-to-peer over mobile ad hoc networks, Wirel. Pers. Commun. 77 (2014) 1167–1189.
[29] H. Sivaraj, G. Gopalakrishnan, Random walk based heuristic algorithms for distributed memory model checking, Electron. Notes Theor. Comput. Sci. 89 (2003) 51–67.
[30] R. Pelánek, T. Hanžl, I. Černá, L. Brim, Enhancing random walk state space exploration, in: Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems, 2005, pp. 98–105.
[31] M.D. Jones, J. Sorber, Parallel search for LTL violations, Int. J. Softw. Tools. Technol. Transfer. 7 (2005) 31–42.
[32] Y. Altshuler, A.M. Bruckstein, Static and expanding grid coverage with ant robots: Complexity results, Theoret. Comput. Sci. 412 (2011) 4661–4674.
[33] R.N. Mantegna, H.E. Stanley, Stochastic process with ultraslow convergence to a Gaussian: The truncated Lévy flight, Phys. Rev. Lett. 73 (1994) 2946–2949.
[34] D.W. Sims, N.E. Humphries, R.W. Bradford, B.D. Bruce, Lévy flight and Brownian search patterns of a free-ranging predator reflect different prey field characteristics, J. Anim. Ecol. 81 (2012) 432–442.
[35] G. Berkolaiko, S. Havlin, Territory covered by N Lévy flights on d-dimensional lattices, Phys. Rev. E 55 (1997) 1395–1400.
[36] F. Bartumeus, E.P. Raposo, G.M. Viswanathan, M.G.E. da Luz, Stochastic optimal foraging: Tuning intensive and extensive dynamics in random searches, PLoS ONE 9 (2014) e106373.
[37] R.E. Walpole, R.H. Myers, S.L. Myers, K. Ye, Probability and Statistics for Engineers and Scientists, vol. 5, Macmillan, New York, 1993.
[38] M.G.E. da Luz, S.V. Buldyrev, S. Havlin, E.P. Raposo, H.E. Stanley, G.M. Viswanathan, Improvements in the statistical approach to random Lévy flight searches, Phys. A 295 (2001) 89–92.
[39] A.S. Ferreira, E.P. Raposo, G.M. Viswanathan, M.G.E. da Luz, The influence of the environment on Lévy random search efficiency: Fractality and memory effects, Phys. A 391 (2012) 3234–3246.
[40] S.N. Majumdar, H. Orland, Effective Langevin equations for constrained stochastic processes, preprint arXiv:1503.02639.
[41] G. Technitis, W. Othman, K. Safi, R. Weibel, From A to B, randomly: a point-to-point random trajectory generator for animal movement, Int. J. Geogr. Inf. Sci. 29 (2015) 912–934.

Please cite this article in press as: M.E. Wosniack, et al., A parallel algorithm for random searches, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.07.014