On the Complexity of Physical Problems and a Swarm Algorithm for k-Clique Search in Physical Graphs

Yaniv Altshuler[1], Arie Matsliah[1], and Ariel Felner[2]

[1] Computer Science Department, Technion, Haifa 32000 Israel
{yanival, ariem}@cs.technion.ac.il
[2] Ben-Gurion University, P.O.B. 653 Beer-Sheva 84105, Israel
felner@bgu.ac.il

Abstract. As the complexity of systems increases, so does the need of examining the nature of *complexity* itself. This work discusses the domain of *physical swarm problems*, in which a swarm of mobile agents is employed for solving physical graph problems (where a certain amount of *travel effort* in required for every movement along the graph's edges). A new kind of complexity scheme, suitable for this domain, is discussed by examining a central problem of this domain — the *physical k-clique* problem. In this problem, a swarm comprising of mobile agents travels along the vertices of a physical graph G, searching for a *clique* of size k. Thus, the complexity of the problem is measured in *travel efforts* (instead of in computation resources). In order to share information between the agents, two communication models are discussed — a complete knowledge sharing (referred to as *centralized shared memory*) and a *distributed shared memory* model, where the mobile agents can store and extract information using the graph's vertices. The work presents a search algorithm for the agents, and discusses its performance under each communication model. The major contribution of this work is demonstrating the strength of the distributed shared memory model. Although this model is much easier to implement and maintain, is highly fault tolerant and has high scalability, the quality of the results it produces is very high, compared to the strongest model of complete knowledge sharing.

Keywords — Physical Graphs, K-Clique, Pattern Matching, Swarm Algorithms, Swarm Intelligence, Distributed Knowledge Sharing

1 Introduction

In recent years, much work has been done in the domain of *physical problems* (or problems in *physical graphs*). Such problems concerns a goal which should be achieved, by one or more mobile agents, which travel along the physical environment. When the group of agents comprises several agents it is also called a *swarm*. The domain of *swarm algorithms* and *swarm intelligence* is also a rapidly growing research field, in which the complexity aspect is of great interest (since usually swarms are assume to comprise very simple agents, with limited resources and capabilities).

Traveling within a physical environment however, is very different than usual information access model assumed in orthodox graph theory, since every movement along an edge of a graph requires a certain amount of *travel effort*. While the travel effort can represents certain amount of time, of fuel, it is immaterial to this model.

However, since movements along the graph no longer assumed to be performed in zero effort, a basic change in the conceptual way in which performance of algorithms is measured, and in which hardness of problem is being formulated, is needed. Hitherto, the complexity of mathematical problems and computer algorithms was mainly measured using the well studied canonical complexity classes such as *P*, *NP*, *P-SPACE*, and others (see [39] for an overview). However, when discussing the domain of physical problems, this scheme should be replaced by some other paradigm, more suitable for the definitions and properties of this domain.

In this work, a new kind of complexity scheme, designed for physical problems, is discussed. Its consequences on the construction of swarm algorithms for physical problems is discussed, and several mechanisms for enhancing the performance of such algorithms, yielding from the complexity scheme are described. This is done by examining a central problem in this domain — the *physical k-clique* problem. In this problem, a swarm of mobile agents travels through a graph G, where its goal is to find a clique of size k (a set of k vertices in which every pair of vertices is connected by an edge in G).

While examining this problem, an algorithm which can be used by a mobile agent is developed, called **PCF** (*Physical Clique Finding*), and its performance examined, as well as several variants of it.

The algorithm is than shown to be an efficient swarm algorithm as well, meaning — be applied successfully in a multi agent environment, where every agent works according to this algorithm. While showing the last, we discusses the calibration of certain parameters of the algorithm, in order to reach optimal performance.

Note that the **PCF** algorithm requires certain cooperation between the agents utilizing it. This cooperation can be achieved by assuming several implicit communication models, using some kind of a shared memory. As an upper bound for the algorithm's performance, we first examine its behavior under the assumption of a *centralized shared memory*, simulating a *completed knowledge sharing* model. As a second step, we examines the algorithm's performance while assuming an allegedly weaker memory model — a *distributed shared memory* model. It is our intention to show that although a system which uses this model is much simpler and easy to implement, the performance which can be achieved over it are not significantly lower than the ones achieved using the superior centralized shared memory model. This is shown using the experimental results in section 7, which demonstrate that while using the proposed algorithm in the distributed shared memory model, the results which are obtained are only slightly inferior than those achieved using a centralized shared memory.

This is surprising since the distributed shared memory model is much simpler to implement and maintain than the centralized shared memory. Moreover, its complexity is far smaller than the stronger model, since its highly scalable — as the graph become larger, small memory units are added to the new vertices. When more agents are added, the communication complexity remains the same, whereas when assuming the centralized shared memory, the communication complexity is $O(k \cdot |V|^2)$.

This fact is perhaps the most meaningful result of this work — although a similar notion was hinted in a few recent works, demonstrating it in a major problem such as the k-clique problem greatly strengthen this notion. This work will be followed by an additional work, currently under preparation, which presents an extended research concerning ideas and suggestions which appear in this work in part.

1.1 Physical Graphs

A *physical graph* denotes a graph $G(V, E)$ in which information regarding its vertices and edges is extracted using *I/O heads*, or *mobile agents*, instead of the "random access extraction" which is usually assumed in graph theory. These agents can physically move between the vertices of V along the edges of E, according to a predefined, or an on-line algorithm or algorithm.

Moving along an edge e, however, require a certain *travel effort* (which might be a constant time, or alternatively, consumes a constant amount of *fuel*). Thus, the complexity of algorithms which work on physical graphs is measured by the total travel efforts required, which equals the number of edges traveled by the agents. We assume that each edge requires exactly one unit of travel effort.

Physical graphs are conveniently used in order to represent many "real world problems", in which the most efficient algorithm is not necessarily the one whose computational complexity is the minimal, but rather one in which the agents travel along the minimal number edges. For example, the *Virtual Path Layout* problem, concerning the finding of a virtual graph of a given diameter, and its embedding in the physical graph such that the maximum load is minimized (see [38] and [40]).

Problems in physical graphs are thus variants of "regular" graph problems, such as finding a *k-clique* in a graph (description and algorithms can be found in [19]), graph and subgraph isomorphism (description and algorithms can be found in [25, 29, 11, 3, 7]), exploration problems (solved for example by *Breadth First Search* (BFS) and *Depth First Search* (DFS) algorithms [23]), etc., whose input graph is a physical graph. Thus, the complexity of these problems is measured as the number of edges an agent (or agents) solving the problem will travel along.

There is a special type of graph problems which can also be ideally described as physical graph problems. Such problems are those in which a *probabilistic*, or *real time* algorithm is used to return a solution which is not necessarily optimal. While a probabilistic algorithm returns a solution which is correct in a probability of $(1 - \epsilon)$ (for as small ϵ as we require), a real time algorithm can be asked at any stage to return a solution, whereas the quality of the solutions provided by the algorithm improves as time passes. Using such probabilistic or real time algorithms, the computational complexity of many problems can often be reduced from exponential to polynomial (albeit we are not guaranteed of finding the optimal solution). Such algorithms can be found for example in [8, 28, 31] (graph isomorphism) and [32, 21, 16] (distributed search algorithms such as *RTA**, *PBA**, *WS_PBA**, *SSC_PBA** and *BSA**). The physical variants of such problems can be thought of as a swarm of mobile agents, traveling the graph and collecting new information during this process. As time advances, more information is gathered by the agents, causing the quality of the solutions provided by the agents to improve.

Notice that while an algorithm which assumes a random access data extraction (from now on be referred to as *random access algorithm*) may read and write to the vertices of G at any order, an algorithm which assumes a physical data extraction (referred to as a *physical algorithm*) must take into account the distance between two sequential operations. The reason for this is that the use of a random access algorithm is performed using a processing unit and random access memory, whereas the use of a physical algorithm is actually done in the physical environment (or a simulated physical environment, which maintain the information access paradigm). Thus, a random access algorithm can access any vertex of the graph in $O(1)$, while a physical algorithm is confined to the distances imposed by the physical metric.

For example, for $u, v \in V$, let us assume that the distance between v and u in G is 5. Then if after a 'read' request from u, the algorithm orders a 'write' request to v, this process will take at least 5 time steps, and will consume at least 5 fuel units. Furthermore, depending on the model assumed for the mobile agents knowledge base, this operation may take even longer, if, for example, the agents are not familiar with the shortest path from u to v, but rather know of a much longer path connecting the two.

As can easily be seen from the previous example, while designing physical algorithms, one must take into account an entire set of considerations which are often disregard, while designing random access algorithms. As a result, algorithms which are designed in order to solve physical problems, may often bear only a few resemblance to ones designed for the corresponding random access problems.

1.2 Communication Models

There are many models for communication in multi agent systems. The most trivial model is of "*complete knowledge sharing*", where any new discovery of an agent is immediately shared with all the other agents. Other models restrict the level of communication. Some models allow broadcasting or message exchanging between agents but restrict the amount of data that can be exchanged in each message or restrict the frequency or the number of messages that are allowed. Many times, a cost is associated with each message and the task is to solve the given problem while trying to minimize the cost of the messages involved.

In nature, ants and other insects communicate and coordinate by leaving trails of odor on the ground. The information placed on the ground is called a *pheromone*. Studies on ants (e.g. [9, 10]) show that the pheromone-based search strategies used by ants in foraging for food in unknown terrains tend to be very efficient. It is believed that ants build a network of information with vertices represented by points of encounter between ants and the information is either passed between ants at a vertex with a physical encounter with other ants or via pheromone traces that are left on the ground.

Inspired by nature, a famous and interesting model for communicating in multi-agent systems is that of ant-walk (e.g. [12, 13, 26]). In this model, information is spread to other agents via *pheromones*, i.e., small amounts of data that are written by an agent at various places in the environment (e.g. edges or vertices in the graph) and can be later used or modified by other agents visiting that vertex.

In our model we study two communication models. The first model assumes a *centralized shared memory*, which can be directly accessed by each mobile agent at any time. Notice that this model is equivalent to the *complete knowledge sharing* mentioned above, which is the most trivial and "strongest" communication model. However, implementing such a model in reality often requires the use of a broadcast capability by the agents, or a fast point to point communication. Thus, there is a strong motivation of presenting other communication model which achieves similar performances to the first.

The second communication model presented in this work is the *distributed shared memory* model. In this model the information network is a graph, and the role of a pheromone is taken by a memory area on each vertex, which each mobile agent can read and modify. The union of these memory areas forms a *distributed shared memory*. Whenever an agent reaches a vertex which contains such memory area, it compares the information located in this memory to the information stored in its database. If the vertex's memory contains new information, the agent incorporates it into its own database, while if the agent had gathered information which is not contained in the vertex's memory, the agent deposits it into the vertex's memory. This process is called a "*data merge*". In this work we assume that all the vertices contain such a memory, although other schemes should be examined as well — requiring only a portion of the vertices to contain such memory module will simplify the implementation of such system and will lower its costs, however the effect of it on the algorithm's performance is yet to be explored.

The distributed shares memory model is especially suitable for large networks (e.g. Internet), in which the vertices are implemented by powerful computers and the edges often have narrow bandwidth and are heavily loaded. This paradigm suggests a distributed group of one or more lightweight autonomous software agents that traverses the network in a completely de-centralized and parallelized manner. Data is spread among the mobile agents via this distributed shared memory. Note that this approach resembles the use of "*large pheromones*", as appears in [42, 41] and elsewhere. However, we believe that the term "distributed shared memory" better describes the nature of the mechanism presented above.

Several works have already been done, considering mobile agents which uses variants of the distributed shared memory model. Such works have shown that such agents are able to cooperate and achieve goals like covering a faulty graph [13], finding an Euler cycle in a graph [26] and solving various combinatorial optimization problems [27].

Following is a communication paradigm between mobile agents assuming a distributed shared memory model in general, and in exploring unknown environments in particular. Each agent has a partial knowledge of the entire environment. It maintains a database with a partial graph that is known to it. Similarly, each vertex holds a database with a partial graph that is 'known' to it, i.e., knowledge that was written to it by the agents. Whenever an agent reaches a vertex, it merges the data known to it with the data that is written in that vertex. The agent then writes the combined data in that vertex and updates its own database according to the new data that was obtained (a *data-merge* operation).

2 Problem

Pattern matching in graphs is the following problem: Given a graph G on n vertices and a graph H on h vertices, find whether G has an induced subgraph isomorphic to H. Many applications of pattern matching can be found in theory and practice of computer science, see e.g. [44] for more details. It is well known that this problem is computationally hard whenever H (and also h) is not constant, but rather a part of the input (a subgraph isomorphism problem), while it has a trivial solution of polynomial complexity if H is a constant graph. Note that the subgraph isomorphism problem is reducible to the *Max-Clique* problem.

This work considers dense physical graphs (graphs which contain many subgraphs isomorphic to the desired pattern) while the goal is to find one of them within as minimal moves on the graph as possible. A graph G on n vertices is called *dense* if the average degree of a vertex in G is $\Omega(n^2)$. Alternatively, we can slightly weaken this condition by requiring that the number of edges in G is $\Omega(n)$, as long as the existence of a large number of the requested patterns can be ensured (see section 7 for more details). The vertices of G are indexed from 1 to n, and the edges of G are indexed from 1 to m, where $m \leq \binom{n}{2}$.

We assume that whenever a vertex $v \in V(G)$ is visited, all edges incident with v are revealed (i.e. their indices are revealed), and naturally v's index is also revealed. Hence, if an edge $e = (u, v)$ exists in G, and some agent visited u and v (in any order), then he can deduce that v is a neighbor of u, even if the agent did not travel on e. If there is a communication between the agents, it is enough that one of the agents visited u and some other agent visited v for this information to be deduced.

One of the artificial examples of a similar model might be an unknown terrain with indexed cities and roads, where the roads are signed with their indices (say with road signs), but their end point indices are not mentioned. However, we do not assume that the graph is planar (i.e. there might be roads, bridges and tunnels, crossing each other in any way).

Similar to ordinary navigation tasks (for example [20–22, 24]), the purpose of each agent employing **PCF** is to reach the "goal vertex" as soon as possible. However, since the goal of the agents is to detect a *k-clique*, the goal vertex is in fact the vertex which when discovered at time t, will complete a *k-clique* in the agent's knowledge base. Thus, there is no specific goal vertex, but rather several *floating goal vertices*, whose identity depends on the following elements :

- The structure of G (namely, V and E).
- The information the agent had collected thus far.
- The information sharing model of the problem (be it a distributed memory, centralized memory, etc').

Note also that this problem is not an ordinary exploration problem (see [4]), where the entire graph should be explored in order for it to be mapped out. Once a requested *k-clique* is found by one of the agents, the problem is terminated successfully. This often occurs while only a small portion of the graph's vertices have been visited.

Another distinction should be made between the problem that is presented in this work and those examined in the field of multi agents routing (see [5, 6]). While multi agents routing mainly deals with the problem of finding paths in dynamic networks where the structure of the environment is dynamically changing due to load balancing and congestion, the *physical k-clique* problem considers a stable physical environment (somewhat similar to the work of [2] and [1]).

3 Motivation

It is our belief that work on swarm algorithms for physical graph problems is strongly required since physical graphs and networks, which take an essential role in everyday's life, are becoming more and more complex. Thus new techniques for such problems must be composed. Several examples for such networks are the world wide web [30, 17, 18], the physical and logical infrastructure of the internet [17], power grids, electronic circuits [17] — all of which are complex physical environments.

The *physical k-clique* problem has several "real world" applications. For example, tracking the connectivity of a computer of telephone network, which can be utilized in order to improve routing schemes within this network. Another example may be a distributed mechanism which search many databases containing transactions and email correspondences, in order to find groups of people who are maintaining a tight communication between them. This information may be used by intelligence units, tracking potential terrorists.

The reason the *physical k-clique* problem was selected for this research is that the *k-clique* problem, on which the *physical k-clique* problem is based, is known to be a significantly hard problem. While most of known NP-complete problems can be approximated quite well in polynomial (sometimes linear) time (as shown in [45]), this is not the case for the *k-clique* problem. An explanation of why there are no "semi-efficient" algorithms for the *k-clique* problem (and that the best solution is exhaustive search) and why the *k-clique* problem can not be approximated in *any* way, unless $P = NP$, can be found in [46]. Additional details regarding NP-Complete problems can be found in [47].

Since the *physical k-Clique* problem is in fact an instance of the *physical pattern matching* problem, solving it can serve as a first step towards a general pattern matching swarm algorithm. In future works we intend to show that the algorithm presented in this work can be applied to any pattern with few modifications.

4 Basic Physical Clique Finding Algorithm

This sections presents the basic swarm algorithm for finding k-cliques in a physical graph. The algorithm described here assumes that the implicit communication used is that of a centralized shared memory (which simulated complete knowledge sharing), meaning — all the agents share the same knowledge base and can perfectly coordinate their operations.

Notice that this model is equivalent to a centralized algorithm in which all the agents are controlled by a "leader" or a "queen", thus it is the strongest model possible, with respects to knowledge sharing and operation coordination.

4.1 Search Algorithm - Definitions and Requirements

Let $r(t) = (\tau_1(t), \tau_2(t), \ldots, \tau_n(t))$ denote the locations of the n agents at time t. The requested search algorithm is therefore a movement rule f such that for every agent i :

$$f\big(\tau_i(t), N(\tau_i(t)), \mathcal{M}_i(t)\big) \in N\big(\tau_i(t)\big)$$

where for a vertex v, $N(v)$ denotes the neighbors of v, e.g. :

$$N(v) \triangleq \{u \in V \mid (v,u) \in E\}$$

$\mathcal{M}_i$ is the memory for agent i, containing information gathered by it through movement along G and by reading information from the shared memory.

The requested rule f should meet the following goals :

- **Finding a k-Clique** : ensuring that :

$$\exists t_{success} \mid k{-}clique \in \mathcal{M}_i(t_{success})$$

  such that this $t_{success}$ is minimal.
- **Agreement on Completion** : within a finite time after the completion of the mission, all the mobile agents must be aware that the mission was completed, and come to a halt.
- **Efficiency** : both in time and space.

The requested rule should also be *fault tolerant*, meaning that even if some of the agents malfunction, the remaining ones will eventually find the target clique, albeit slower.

4.2 The PCF (Physical Clique Finding) Algorithm

For solving the *Physical k-Clique* problem we suggest the **PCF** search algorithm. This algorithm can be described as follows. Each agent holds a knowledge base which contains some information regarding G. Every time an agent enters a vertex v, it performs a data merge process, in which the knowledge base of the agent updates and is updated by the central knowledge base.

The main idea in the search algorithm is exploring the graph in directions that have the highest potential of discovering the desired pattern. In our case, considering only cliques simplifies the arguments because of their perfect symmetry, but it still emphasizes the main ideas of the general case — a generic search algorithm for any given pattern.

This is done by sorting the potential sets of vertices according to the largest clique which is contained in the set. Within the same size of largest clique, the sets are sorted according to the total number of unexplored edges which touch the vertices of the set (unexplored edges are edges $e(v, u)$ whereas at least one the identities of v and u is yet unknown) . As large the number of such edges is, the more likely it is for the set to be expandable to a *k-clique*. In addition, if a *k-clique* was found, the sort criteria will place it on the top of the list, and hence the agents will immediately recognize it. To maintain the lists compact, the algorithm focuses on certain sets (the top most in the sorted list) and either eliminates them or discovers that they are expandable to a *k-clique*.

Our algorithm uses a *job list*, where each job is associated with a set of vertices that are potentially expandable to a *k-cliques*. The jobs are sorted primarily by the size of their corresponding sets, and secondarily by the number of unexplored edges incident to one of the vertices in the set. In addition to the set of vertices, a job holds an instruction to the agent that takes it. The instruction is of the form "travel on edge e from vertex s", where $s \in V$ is the source vertex, and the agent might need to go to s before he performs the instruction of the job.

Generally, when looking for a pattern H of size h in graph G, every set of $m, m < h$ visited vertices in G that might be completed to a subgraph of G isomorphic to H (i.e. there are enough unexplored edges for every vertex) forms a potential sub-H of size m. While considering cliques as the patterns, we only need to verify that the m vertices of the set form a clique and that every vertex in the set has at least $h-(m+1)$ unexplored edges (which is the minimal requirement in order for this set to be expendable to an $h - clique$.

The list of jobs is updated (if needed) after every move of the agents.

If there are α available agents in a turn, the algorithm assigns the first α jobs in the sorted list to the agents, where the assignments are made in a way that minimizes the total travel distance in L_∞ norm. This is done in order to minimize the travel efforts of the agents. This is an example of a difference between physical problems and "regular" problems — whereas in conventional complexity scheme all the ways of dispersing the jobs among the agents are identical, in the complexity scheme of physical problems we would like to assign jobs to the nearest agents.

Once an agent reaches its goal (the closest vertex of the job assigned to the agent), it writes the new discovered information in the adjacency matrix and becomes available. In the beginning of each turn, if the first job in the list is associated to a list of k vertices the algorithm declares that a clique is found and terminates.

Notice that all the agents use a common job list. In addition, the agents are assumed to broadcast every new information they encounter.

We assume that all distances (i.e. graph edges) are equal to one unit, and that the agents have sufficient memory and computational power. We also assume that that there are many cliques of the required size in G (otherwise, if the number of cliques is too small (e.g. $o\binom{n}{k}$), the optimal algorithm for discovering them would be the exhaustive exploration of G).

The algorithm, used by each agent i appears in figure 1.

4.3 Correctness

The main observation, on which our algorithm is based, is that none of the agents travel on same edge more than once. Hence, even in the worst case scenario, the whole graph will be explored within no more than $|E|$ moves (which is asymptotically similar to the standard *DFS* and *BFS* algorithms). Therefore, our algorithm terminates after at most $O(|E|)$ moves. Since all *k-cliques* are in the top of the job list in some stage, the algorithm described above eventually recognizes them.

```
Algorithm **PCF** :
While the first job in the common job list is not associated to a set of $k$ vertices
    Assign top job in jobs list to the agent;
    Perform the job;
    Update the agent's data structures;
    Update the agent's jobs list (or the common list);
    Update the list of available agents;
    If (all vertices explored) then
        **STOP**;
End **PCF**;

Procedure **CREATE DATA STRUCTURES**$(i)$ :
Create an $n \times n$ matrix with entries of type integer;
/* Represents edge indices according to vertex adjacency of G, as discovered by the agents. */
Create a sorted list of jobs;
/*
Each job is associated with one potential clique. The job list is sorted according to the following criteria :
(a) Size of the potential sub clique
(b) Number of still unexplored edges
(c) The distance of the job from the agents
The distance is computed according to $L_\infty$ norm, that is the min—max on the travel distance.
*/
End **CREATE DATA STRUCTURES**;

Procedure **INITIALIZE**$()$ :
Initialize the agents' jobs lists;
/*
Note that the common job list may contain large potential sub-cliques, immediately upon initialization.
*/
**CREATE DATA STRUCTURES**$(i)$;
Choose random starting points on $G$ for the agents;
Place the agents in these starting points;
Start the agents according to the **PCF** algorithm;
End **INITIALIZE**;
```

Fig. 1. The **PCF** search algorithm for the centralized shared memory model.

5 Physical Clique Finding Algorithm — Distributed Shared Memory

This sections presents the variant of the basic swarm algorithm for finding k-cliques in a physical graph, while assuming the distributed shared memory model.

Notice that the algorithm is *fault tolerant*, meaning that even if some of memory units within the graph's vertices will malfunction, the swarm will still be able to function and complete its mission with only a slight decrease of efficiency.

The general principles of the algorithm are identical to those appear in section 4.2 although the following features were added.

Every time an agent enters a vertex v, the data merge process is performed between the knowledge bases of the agents and this stored within v.

Notice that while in the centralized shared memory model, all the agents use a common job list, in the distributed shared memory model, each agent is equipped with its own list.

Since the agents are allowed to leave data in the vertices of the graph — we assume without loss of generality that the information stored will include all their knowledge base at the time of arrival to this vertex. This is comparable to the assumption made in the centralized shared memory model that the agents broadcast every new information they encounter.

The correctness of the algorithm is derived from the correctness of the centralized **PCF** algorithm, as appears in section 4.3.

The algorithm, used by each agent i appears in figure 2.

5.1 Guaranteeing Operation Diversity

When introducing to a physical graph more than one agent, using the **PCF** algorithm, and assuming a distributed shared memory, one may discover that several agents may travel together, causing the performance of the system to degrade. This problem is caused since all agents follow the same deterministic algorithm, and are initially located at the same vertex. Thus, all agents face the same initial information regarding the graph. As a result, they will necessarily make the same decisions and follow the same path. When m out of n agents visit the same vertices at the same time, for all practical purposes, the performance achieved are equivalent to those achieved when using only $(n - m)$ agents.

In order to avoid this problem, several techniques may be used. First, we may use a disperser protocol, which will choose random values for the first m moves of the agents, thus guaranteeing (with a high probability) that in time step m, no two agents will be located in the same vertex. Thus, the agents will now be able to start activating the **PCF** algorithm.

Another method which could be used is "breaking the symmetry" of agents entering the same vertex. At any time during the operation of the agents, two or more agents may enter the same vertex at the same time. In order to prevent such agents from "uniting" and start traveling together, a mechanism which will force an asymmetry between the agents must be implemented. By using such a mechanism, the agents could be able

```
Algorithm **PCF** :
While the first job in the agent's list is not associated to a set of $k$ vertices
    Assign top job in jobs list to the agent;
    Perform the job;
    Update the agent's data structures;
    Update the agent's jobs list;
    Update the agent's availability status;
    If (all vertices explored) then
        STOP;
End PCF;
----------------------------------------------------------------
Procedure CREATE DATA STRUCTURES($i$) :
Create an $n \times n$ matrix with entries of type integer;
/* Represents edge indices according to vertex adjacency of G, as discovered by the agents. */
Create a sorted list of jobs;
/*
Each job is associated with one potential clique. The jobs list is sorted according to the following criteria :
(a) Size of the potential sub clique
(b) Number of still unexplored edges
(c) The distance of the job from the agent
*/
End CREATE DATA STRUCTURES;
----------------------------------------------------------------
Procedure INITIALIZE() :
Initialize the agents' jobs lists;
*/
Note that each list will be initialized to contained at most a potential singleton (the vertex the agent is currently located in).
*/
CREATE DATA STRUCTURES($i$);
Choose random starting points on $G$ for the agents;
Place the agents in these starting points;
Start the agents according to the PCF algorithm;
End INITIALIZE;
```

Fig. 2. The **PCF** search algorithm for the distributed shared memory model.

to generate a local order of the agents which are located in the same vertex, and use this sorting in order to ensure that the problem above will not occur. Since the **PCF** algorithm uses “job queues”, and since when two agents are entering the same vertex, their knowledge base, and thus also their job queues, must be identical. By using the local order each agent can select the i-th job in its queue (rather than the top job), when i denote its place in the local order. Thus, we guarantee that each agent will select a different destination.

A sorting mechanism, as mentioned in the previous paragraph, can be implemented using the IDs of the graph’s edges E. Since we assume each edge (v,u) has a distinct ID, known to agents arriving either from v or from u, this ID can be used in order to create the requested sorting. Let $e_{i,v}$ be the edge through which agent i had entered the vertex v. Thus, all the agents which had entered v examines $ID(e_{i,v})$ for every agents i in v. The IDs of the edges are than sorted (locally, by each agent), and through this sorting, the agents in v are sorted, when each agent i places itself according to the position of $ID(e_{i,v})$ in the ordered IDs of edges through which the agents had entered v. Since the system is not assumed to be synchronized, if two agents are entering v using the same edge e, one of them does this earlier, and thus it will be placed before the rest of the agents which entered v through e.

This problem, of course, does not appear when assuming a centralized shared memory. This is so since the centralized memory model also encapsulates the mechanism which enables each agent to “pop” a task from the global queue, with no chance of other agents selecting the same destination.

The problem of preventing simple agents from performing the same movements and thus decrease the swarm’s total performance was discussed in various works which presented multi agents or swarms based systems. Since each problem assumes a different model, both for the environment in which the agents operate and for the capabilities and limitations of the agents, a different mechanism of avoiding this undesired phenomenon must be produced for each problem. For example, see [41, 35, 36].

6 PCF Algorithm — Improvements and Extensions

6.1 Towards a Generic Physical Patterns Finding Algorithm

The main difference between cliques and graph patterns in general is that cliques (or alternatively independent sets) have a perfect symmetry between their vertices. In other words, every pair of vertices in a clique has equivalent elements under some automorphism.

The notion of “potential” is also naturally extensible to the general case. For example, given a pattern H on h vertices and a subset C of c vertices from the graph G, we say that C is potentially expandable to H if :

1. $c < h$.
2. There exists an assignment π to the unknown pairs of vertices.
3. There exists a set C' of $(h-c)$ vertices such that the induced subgraph on $C \cup C'$ under the assignment π is isomorphic to H.

To scale the potential of a certain subset C, we compute the amount of possible assignments under which C is expandable to a set of vertices that induce a subgraph isomorphic to H.

6.2 Communication Agents

In order to further improve the performance of the **PCF** algorithm, working in the distributed shared memory model, the notion of *communication agents* was introduced. Communication agents are agents which instead of following the top task of the queue, decides to chose an arbitrary destination, and move towards it. Whenever such agents arrive to a new vertex, they perform the data merge process, similar to regular agents (referred to as *searching agents*). As discussed earlier, the goal of the searching agents is to gather new data regarding vertices whose potential of being a part of a large clique is high. The goal of the communication agents, however, is quite different — since searching agents only follow their tasks queue, there is a danger that the information collected by some of these agents will not become available to other searching agents, for which it might be extremely relevant. Thus, the random nature of the communication agents' movements is aimed for making sure information gathered by the searching agents proliferate along the graph.

Since the primary task of the communication agents is to spread data and thus perform as many data merge operations as possible, a communication agent may choose the vertex where the contribution of the data merge operation will be as significant as possible, as its destination. Furthermore, when such an agent arrives at a new unexplored vertex, this vertex is added to the general knowledge of the system (immediately in a centralized shared memory model, and partially in the distributed shared memory model).

There arises the question of the exact mechanism which will instruct an agent to become a communication agent. Notice that the simplest mechanism will chose a portion of the agents, upon initialization, and mark them as communication agents. However, there is also the possibility of dynamically allocating communication tasks to agents — whenever an agent will become available for a new job, it could be assign a communication job in some probability p, or a (regular) searching task in probability $(1 - p)$. Notice that this allocation will assign one job only — when this agents will become available again, it could be assigned either communication or searching task.

Note that if the dynamic allocation method is used, there exist again a wide range of possibilities for deciding the probability p — should it remain constant throughout the entire mission, or should it change (increasing or decreasing the number of communication agents at any given time) ? Should it depend on the parameters of the environment (the size of the requested clique, the size of the graph, etc.) ?, should it depend on the parameters of the swarm (the number of agents, other enhance mechanisms which are used, etc.) ?

Let p_{comm} denote the probability for an agent to being assigned a communication mission. In order to answer these questions and find the optimal scheme in which the communication agents mechanism should be used, the following alternatives were implemented and tested :

- p_{comm} was first defined as a static number, upon the initialization of the system. This means that the at any given time, the number of agents which served as communication agents was roughly the same. When implementing this alternative, many values for p_{comm} were tested, in order to find the optimal value for this parameter.
- After simulating with static values for p_{comm}, as dynamic scheme was examined, in which the value of p_{comm} changes in time. The value of p_{comm} is defined according to the *simulated annealing* model (see for example [33]), where in each time step t the system has a temperature C_t which represents the "noise" of the system — the chance that an agent will perform a random movement. The temperature of the system changes according to a certain rule :

  $$p_{comm} = e^{-\frac{t}{C_0}}$$

  when C_0 is the initial temperature of the system.
- Similarly to the previous method, the value of p_{comm} is dynamic, where in p_{comm} itself is a function, defined as follows :

  $$p_{comm} = e^{-\frac{t \cdot f}{C_0}}$$

  while f is the utility of the last movement (calculated according to the increase or decrease of the potential of the top tasks in the tasks queue). Thus, when an agent had performed a movement which had improved its knowledge base (for example, by adding vertices to the largest clique known to the agent), it will have much lower probability of becoming a communication agent, than an agent whose last movement yielded no new valuable information.
- Alternative policies for the communication agents should be developed and examined. Since performing a random move does not necessarily guarantee that the agents will reach their goal — spreading the information among the swarm — a more complex *communication movement algorithm* is required. Such algorithm may take advantage of information regarding the positions of the agents themselves, and not only information concerning the graph. This will allow communication agents to ensure the connectivity of the swarm more efficiently.

6.3 Exploratory Agents

Similar to the use of communication agents in the distributed shared memory model, there is also a possibility of using a special kind of agents, in order to enhance the **PCF** algorithm, in the centralized shared memory. Such agents are called *exploratory agents*.

The goal of the exploratory agents is to try and add robustness to the basic algorithm used. Since the main principle of the **PCF** algorithm is sorting all the known sub-cliques according to their size, unexplored edges and proximity to the agents, it can easily be seen that in some "adversary generated" cases, a swarm of agents utilizing this algorithm will face much difficulties in finding a requested $k - clique$. In fact, such swarm may complete the mission even slower than a swarm of agents which decide their next movements randomly ! It is true that such cases are rare, however, a method of preventing them (or at least — restraining their effect) is required.

The allocation of a portion of the agents as exploratory agents achieves exactly that — by following the very simple rule of "*selecting the next goal vertex randomly*", the agents are immune to cases in which the logic of the **PCF** algorithm fall into some corner-case of the problem topology. Thus, the use of exploratory agents adds the needed robustness to the system, albeit slightly decreasing its performance, since less agents are now using the **PCF** algorithm.

When examining the issue of exploratory agents many questions arise, similar to those which concerns the case of communication agents (see section6.2). Should the identity of the exploratory agents be decided upon initialization or should it use dynamic allocation ? Should the number of exploratory agents remain constant throughout the operation of the agents, or should it change (in which case, should it increase or decrease, as time advances ?).

Similar to the case of communication agents, many techniques have been tested, trying to find the best scheme for the use of exploratory agents. The results in section7 provide some answers as to the method of use.

6.4 Meta Data

Throughout this work, we assume that the agents use the shared memory (be it centralized or distributed) in order to store only information which concerns the graph's structure. However, allowing the agents to store other kinds of information, may dramatically improve their search performance, since the essential element which stands in the basis of all swarm algorithms is the (mostly implicit) cooperation between the agents. Thus, storing and using "*swarm meta-data*", which describes the past movements and even future movements which the agents intend to perform, can greatly increase this cooperation and as a result, create a more efficient and organized swarm.

Such meta data may include for example a time stamp for each piece of information deposited into the shared memory. Such time stamps enable the reconstruction of the recent movements the swarm's agents had performed (when "fresh" time stamps indicate a presence of agents in this region, lately).

Apart from examining what kind of meta data can be produced and stored in the shared memory, the specific nature of utilizing this data by the agents should also be investigated. Hitherto, we have not yet fully explored the use of meta-data in the *physical k-clique* problem, although we intend to present such results in future works.

7 Results

The algorithm was tested on *Erdös-Renyi* random graphs $G \sim G(n, p)$ where G has n vertices, and each pair of vertices form an edge in G with probability p independently of each other. In order to ensure that G has enough clique sub-graphs of size k, the value of p should be chosen wisely. Formally, the probability that k vertices of G form a clique is p^k, thus by linearity of expectation and the second moment, G will have at least $\frac{1}{4}\binom{n}{k}p^k$ cliques of size k with probability $1 - o(1)$ (for further background on probabilistic arguments see e.g. [43]). Hence we choose the value of p with respect to this formula.

It is essential that the number of desired cliques in the physical graph is of adequate order (meaning — high enough). If, for example, the number of cliques in the graph is order of unity, then the considered problem becomes trivial, since solving it in the physical environment requires nearly full exploration of the graph. This will be the case for a great majority of the problem instances, in view of the fact that we assume the initial placement of the agents on the physical graph to be chosen uniformly at random. Since environment exploration and similar problems were already studied (for example, in [4]), the relevance of such sparse graphs is rather low.

Furthermore, once a graph is fully explored, every classical algorithm might be applied for finding cliques in it. Thus, in some sense the difference between the physical environment which is almost fully explored, and the classical random access model is diminished. Hence the efficiency of algorithms in physical environment should be naturally measured on problems that do not inherently require exploration of the graphs (or in other words, problems that can be decided after acquiring only a partial information on the graph).

Figure 3 contains experimental results of the search algorithm for cliques of size 10 in graphs of sizes 500 and 2000. The number of agents searching the graphs is 5 through 50 agents. The communication model assumed is the centralized shared memory. It can be seen that while adding more agents dramatically improves the search time, the system reach a saturation at some level, after which adding more agents yield a mild contribution to the performance, if any. This can be best observed in the smaller graph, in which enlarging the number of agents from 15 to 50 decreases the search time by less than 30%, where as increasing the number of agents from 5 to 15 decreases the search time by 70%.

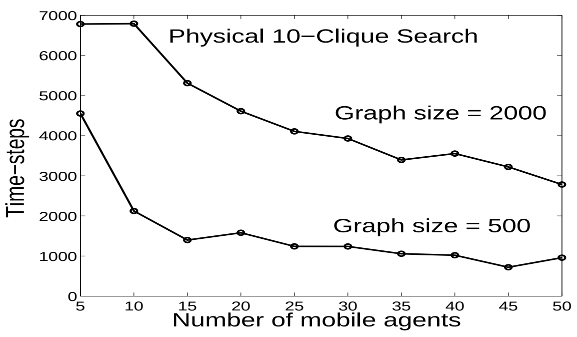

Fig. 3. Results of the Physical 10-Clique problem, with a centralized shared memory model. Notice how the performance of the swarm increases while adding more agents, until it reaches a saturation.

Figure 4 examines the increase in the swarm's search time for larger graphs. Notice that unlike the regular *k-clique* problem, in which the search time grows exponentially with the input graph's size, since we demanded that the number of cliques will be proportional to the graph's size, the search time should increase moderately, or does not increase at all (depending on the exact edge probability used while building the graphs). Note that the search time for 20 agents is consistently smaller than for 10 agents. However, while in smaller graphs the difference is approximately 50%, for larger graphs it shrinks down to 25% for a graph of 4500 vertices. Interestingly, when increasing the number of agents to 30, the performance almost do not improve at all (meaning that the system had reached a state of saturation).

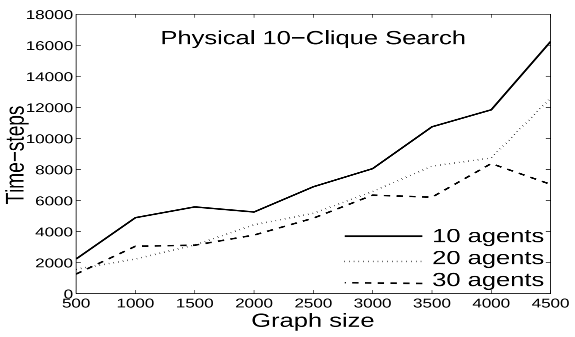

Fig. 4. Results of the Physical 10-Clique problem, with a centralized shared memory model. Notice how the performance of the swarm increases while increasing the number of agents from 20 to 30, whereas this number is increased to 30, the performance almost never changed.

Figure 5 presents the improvement in performance of a swarm working within the distributed shared memory model, as the number of agents increases. Notice that the improvements rate is much faster that in the centralized shared memory model. Figure 6 demonstrates how the performance of the distributed model improves faster than those of the centralized model.

Figure 7 demonstrates how the performance of the distributed model are only about 180% than those of the centralized model. This is most important, since one might suspect that replacing the strongest model possible (this of complete knowledge sharing) with a much simpler one (of distributed memory) might cause a dramatic decrease in performance.

Figure 8 contains a comparison between the centralized model, the distributed model, and the distributed model where 10% of the agents are assigned to be communication agents. Notice that the results show that (at least in this case) adding the communication agents slightly decrease the system's performance.

Figure 9 examines the change in performance in comparison to the size of the input graph. Results for both centralized and distributed models are presented, where an interesting observation can be made — while the performance of the centralized model in the smaller graphs is superior to those of the distributed model, the distributed model is much less vulnerable to the increase in the vertices number. This can be observed as the size of the input graph increases — in the case of the distributed model. the perfor-

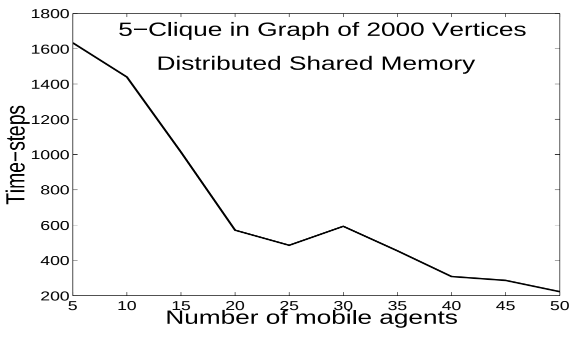

Fig. 5. Results of the Physical 5-Clique problem, in a graph of 2000 vertices, using the distributed shared memory model. Notice the fast improvement of performance as the size of swarm increases.

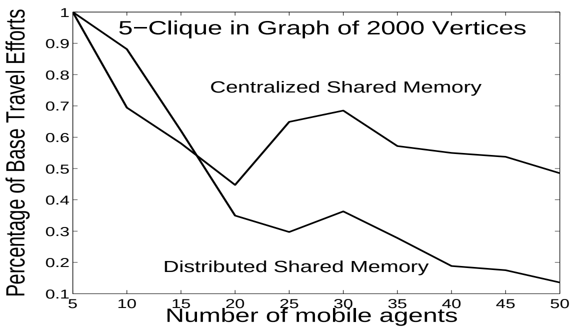

Fig. 6. Results of the Physical 5-Clique problem, in a graph of 2000 vertices — a comparison between the centralized and the distributed models. The data represents the travel efforts as percentage of the travel efforts for 5 agents. Notice how the performance of the distributed model improves faster than those of the centralized model.

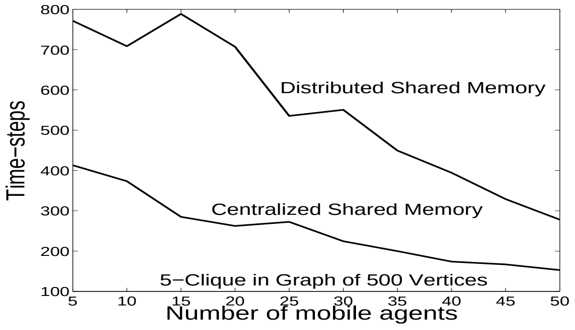

Fig. 7. Results of the Physical 5-Clique problem, in a graph of 500 vertices — a comparison between the centralized and the distributed models. Notice how the performance of the distributed model approaches slowly those of the centralized model. In average, a performance using the distributed model are only 180% than those achieved using the centralized model.

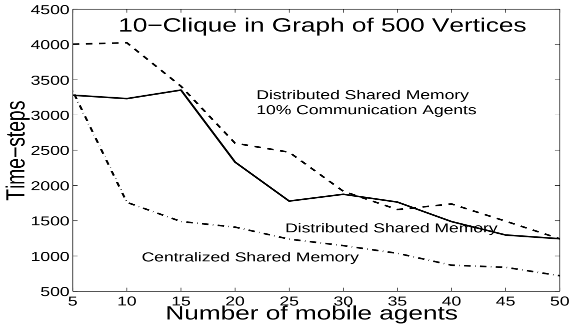

Fig. 8. Results of the Physical 10-Clique problem, in a graph of 500 vertices — a comparison between a centralized, a distributed model and a distributed model with 10% of the agents assigned as *communication agents*.

mance for a graph of 4500 vertices are only 88% more than those of 500 vertices. As the same time, the travel effort for the centralized model increases by 367%.

The reason for this remarkable phenomenon may be the lack of scalability of the centralized model, which may cause its performance to degrade in larger graphs.

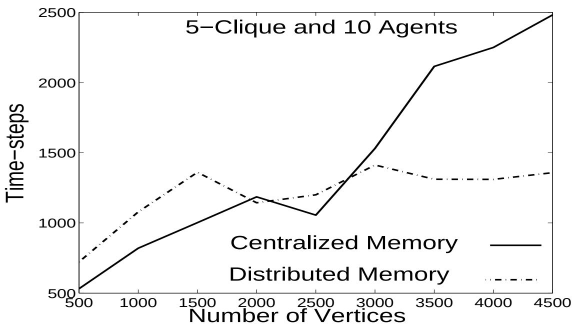

Fig. 9. Results of the Physical 5-Clique problem, in graph of growing number of vertices — a comparison between the centralized memory model and the distributed model. Note that while the performance of the centralized model in the smaller graphs is superior to those of the distributed model, the distributed model is much less vulnerable to the increase in the vertices number — the performance for a graph of 4500 vertices are only 88% more than those of 500 vertices whereas for the centralized model, the performance for a graph of 4500 vertices are 367% more than those of 500 vertices.

8 Observations

As expected, increasing the size of the swarm always improve the swarm's performance. However, there is a state of saturation to which the system can arrive, after which adding more agents will achieve only mild improvements, if any. As small the graph is, the less agents can be added before the system is saturated.

Note that while adding agents to a swarm operating in the centralized memory model do increase its performance, the effect of the same addition of agents on a swarm which operates in a distributed memory model is significantly noticeable.

Regarding the mechanism of communication agents — so far no contribution of such agents were shown, although it may be due to naive communication algorithm, which should be replaced for a more sophisticated one.

Another interesting observation is that unlike orthodox complexity scheme, used for "regular *k-clique* problem", in which the search time grows exponentially with the input graph's size, since we demanded that the number of cliques will be proportional to the graph's size, the search time of the swarms increases only moderately. This is yet another fascinating basic difference between the two complexity schemes.

Furthermore, it was discovered that while the centralized model in more affected by the increase in the graph's size, the effect of it on the distributed model is much more moderate. In fact, when working in relatively large graphs (4500 vertices) the performance of the distributed model were sometimes superior to those of the centralized model. This can be explained by flaws in the centralized algorithm. Such flaws may be inherent in the entire approach of such centralized algorithms, which leaves no room for stochastic influences by the environment, which may help direct the search towards the requested patterns.

This should be combined with the fact that almost in all simulations the performance of the distributed model was always close to this of the centralized model, while the rate of improvement of the distributed model was faster (meaning that for larger swarms and for larger graphs the performance of the two models would become more and more similar). Together, these two observation help demonstrate the basic hypothesis of this work, which is that while the distributed may appear at first simple and limited, it's strength is actually not far from this of the strongest knowledge sharing model possible. Taking into account the scalability, fault tolerance and the simplicity of the implementation of this memory sharing model, emphasizes its advantages even more.

9 Related Work

Hitherto, there have been various works which examine problems in physical graphs, as well as works which uses a distributed shared memory of some sort, have been performed. For example, [6, 14] use mobile agents in order to find shortest paths in (dynamically changing) telecom networks, where the load balancing on the edges of the graph is unstable. Similarly, several widely used Internet routing algorithms (such as BGP [34], RIP [37] etc') propagate shortest path information around the network and cache it at local vertices by storing routing tables with information about the next hop. Another known routing system which uses "ants" and "pheromonoes" is the *AntNet* system [5]. In AntNet, ants randomly move around the network and modify the routing tables to be as accurate as possible.

While these approaches seem similar, there is a great difference between these works and one presented here, both concerning the environment in which the agent operate, the data that is stored at each vertex and the problem to be solved.

First, most of the works mentioned above which concern routing problems, assume that the environment is a telecom network of some sort, which changes dynamically over time. In this work, we assume the graph to be fixed and corresponds to a real physical environment (with Euclidean distances or some other metric), while the difficulty is derived from fact that the graph is not known to the agents.

Another difference is that these algorithms try to help a mobile agent (which for example, represents a phone conversation) to navigate to a certain goal. In other words, given the destination vertex (which could be any vertex of the environment) and the knowledge written in the current vertex, these algorithms try to locally decide where should the agent go to next, while the goal of each agent is merely to minimize the time it takes this particular agent to reach its destination. In the *physical k-clique* problem, on the other hand, the agents' goal is to find k vertices which form a *k-clique*, while the goal of the agent is to minimize the time it takes the entire *swarm* to find such a clique.

Third, when dynamic graphs are concerned, the accuracy of the data stored within the vertices often degrade over time. Thus, while designing algorithms for dynamic physical graphs, much effort is being put on keeping the data as accurate as possible at all times. This is often achieved by having ensuring that the mobile agents consistently update this information. In such paradigm, the accuracy of the stored data must be analyzed by the agents, which must decide whether this data is still relevant. Since our problem assumes that the data placed at a vertex stays valid at all times, the algorithm presented contain several basic differences from the ones mentioned above.

Essentially, the approach presented in this work can be seen as a generalization of the *next hop lookup tables* mentioned earlier, since according to the partial graph information stored in each vertex, an agent decides on its next move, when this process is continually advancing, until a clique is found.

Similar to this approach, the work of [15] also employed a distributed shared memory (a *distributed blackboard*, or a *distributed database*), while the works of [41] and [42] present a mechanism which find the shortest path within a physical graph, while assuming several communication mechanism which are similar to the concepts which appear in this work.

10 Summary and Future Work

This work had presented the *physical k-clique* problem, where a swarm comprising n mobile agents travels along the vertices of a physical graph, searching for a *clique* of size k. In order to share information between the agents, two communication models were examined. First, a full communication model is assumed, referred to as *centralized shared memory*. Than, a *distributed shared memory* model is examined, which enables the mobile agents to store and extract information using the graph's vertices.

The **PCF** search algorithm, which can be employed by a swarm of mobile agents traveling along the graph's edges, was presented. The algorithm was discussed and experimental results for it were shown.

The results demonstrated that the distributed shared memory model can produce results which are very close to those achieved by using the much stronger centralized shared memory model.

While examining the *physical k-clique* problem, new interesting opportunities for an extended research have emerged. We have already started investigating some of the above, producing more valuable results. Following are various aspects of this ongoing and future research :

1. The current version of the **PCF** algorithm was designed for static physical graphs. The problem of clique finding in dynamic physical graphs, however, bears a great deal of interest. The dynamic version of this problem would concern a graph G which in each time step t, each pair of its vertices u and v such that $\neq ((u, v) \in E_t)$ has a probability p_{on} for $(u, v) \in E_{t+1}$ and each pair v, u such that $(u, v) \in E_t$ has a probability p_{off} for $\neg((u, v) \in E_{t+1})$. Such a problem has many interesting applications as well.

2. As mentioned in previous sections, although the algorithm described in this work was designed in order to find *k-cliques* in a physical graph, we believe that by minor adjustments a generic algorithm for finding a given pattern in a physical graph may be composed. We have starting examining this issue, in hope of presenting such an algorithm in the future.

3. Since the algorithm was designed to be employed by a swarm of mobile agents, it is highly dependent on the proliferation of information (performed by the agents by the data merge processes). One method used in order to enhance this aspect of the algorithm, was the introduction of the communication agents to the system (see section 6.2. However, since this aspect significantly affects the performance of the algorithm, an analysis of the information proliferation process should be performed. Among others, we hope to achieve bounds for the rate in which information is distributed in graphs, as well as for the optimal percentage of communication agents and correlation of this value and for the features of the input graph.

4. As important the use of communication agents will become, more emphasis should be put no designing "smarter" and more efficient agents of this type. More complex heuristics ought to be developed in order to accelerate the performance of the communication agents in gathering the maximal amount of information and distributing it optimally (by estimating where and when it will be sought for).

5. The same goes for exploratory agents, which are used in the centralized shared memory model.

6. In addition to the development of better ways of using the communication and exploratory agents, more research is needed considering the best mechanism of controlling p_{comm} and $p_{explore}$ — the probabilities of becoming such an agent. Extensive search for the optimal parameters defined in sections 6.2 and 6.3 is required, as well as developing new methods of dynamically changing these probabilities.

7. Apart from performing simulations for the algorithm, and in order to fully understand the difficulty of the problem and the performance of the algorithms developed for it, analytic bounds should become available. to correctly appreciate the performance of algorithms for the problem, a lower bound for the shortest search time for a *k-clique*, for a given input graph, should be developed. Such a bound will help put into the right perspective empirical results of the simulations, since it will not be algorithm dependent. On the other hand, upper bounds for the search time for the specific algorithm used should be developed as well, in order to guarantee certain minimal performance of those algorithms.

8. An additional research should also be performed concerning the domain itself — its features and topology. Such research can help us to better foretell for example whether the use of communication and exploratory agents is expected to improve the swarm's performance. Such research can also help us design better variants of the communication and exploratory algorithms, as well as of the **PCF** algorithm itself.

9. The use of dynamic data, as suggested in section 6.4, should be studied.

References

1. I.A. Wagner, M. Lindenbaum, A.M. Bruckstein: “Efficiently Searching a Graph by a Smell-Oriented Vertex Process”, Annals of Mathematics and Artificial Intelligence, Issue 24 (1998), pp. 211–223
2. R.C. Arkin, T. Balch: “Cooperative Multi Agent Robotic Systems”, Artificial Intelligence and Mobile Robots, MIT/AAAI Press, Cambridge, MA (1998)
3. B.D. McKay: “Practical Graph Isomorphism”, Congressus Numerantium, 30, pp. 45–87, (1981).
4. M.A. Bender, A. Fernandez, D. Ron, A. Sahai, S.P. Vadhan: “The power of a pebble: Exploring and mapping directed graphs”, In the proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computating, pp. 269–278, Dallas, Texas, May (1998).
5. G. Di Caro, M. Dorigo: “AntNet:Distributed stigmergetic control for communiction networks”, Journal of Artificial Intelligence Research, 9:317–365, (1998).
6. S. Appleby, S. Steward: “Mobile software agents for control in telecommunication networks”, British Telecom Technology Journal, 12, pp. 104–113, (1994).
7. D.G. Corneil, C.C. Gotlieb: “An efficient algorithm for graph isomorphism”, Journal of the Association for Computing Machinery, 17, pp. 51–64, (1970).
8. W.J. Christmas, J. Kittler, M. Petrou: “Structural Matching in Computer Vision Using Probabilistic Relaxation”, IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 17, no. 8, pp. 749–764, (1995).
9. F.R. Adler, D.M. Gordon: “Information collection and spread by networks of partolling agents”, The American Naturalist, 140(3):373–400, (1992).
10. D.M. Gordon: “The expandable network of ant exploration”, Animal Behaviour, 50:372–378, (1995).
11. M.R. Garey, D.S. Johnson: “Computers and Intractability: A Guide to the Theory of NPCompleteness”, Freeman & co., New York, (1979).
12. I.A. Wagner, A.M. Bruckstein: “ANTS: agents, networks, trees and subgraphs”, Future Generation Computer Computer Systems Journal, 16(8):915–926, 2000.
13. V. Yanovski, I.A. Wagner, A.M. Bruckstein: “Vertex-ants-walk: a robust method for efficient exploration of faulty graphs. Annals of Mathematics and Artificial Intelligence, 31(1–4):99–112, (2001).
14. R. Schnooderwoerd, O. Holland, J. Bruten, L. Rothkrantz: “Ant-based load balancing in telecommunication networks”, Adaptive Behavior 5(2), (1996).
15. P. Valckenaers, V. Marik, D.C. McFarlane: “Holonic and multi-agent systems for manufacturing”, First International Conference on Industrial Applications of Holonic and Multi-Agent Systems, (1993).
16. J.B.H. Kwa: “BS*: An Admissible Bidirectional Staged Heuristic Search Algorithm”, Artificial Intelligence, pp. 95–109, Mar., (1989).
17. S.N. Dorogovtsev, J.F.F. Mendes: “Evolution of networks”, Adv. Phys. 51, 1079, (2002).
18. D.J. Watts: “Small Worlds”, Princeton University Press, Princeton NJ, (1999).
19. D.S. Hochbaum, O. Goldschmidt, C. Hurken, G. Yu: “Approximation algorithms for the k-Clique Covering Problem”, SIAM J. of Discrete Math, Vol 9:3, pp. 492–509, August, (1996).
20. P. Cucka, N.S. Netanyahu, A. Rosenfeld: “Learning in navigation: Goal finding in graphs”, International journal of pattern recognition and artificial intelligence, 10(5):429–446, (1996).
21. R.E. Korf: “Real time heuristic search”, Artificial intelligence, 42(3):189–211, (1990).
22. L. Shmoulian, E. Rimon: “Roadmap A*: an algoritm for minimizing travel effort in sensor based mobile robot navigation”, In the proceedings of the IEEE International Conference on Robotics and Automation, pp. 356–362, Leuven, Belgium, May (1998).
23. R.J. Wilson: “Introduction to Graph Theory”, Longman, London, 2nd ed., (1979).

24. A. Stentz: "Optimal and efficient path planning for partially known environments.", In the proceedings of the IEEE International Conference on Robotics and Automation, pp. 3310–3317, San Diego, CA, May (1994).
25. L.P. Cordella, P. Foggia, C. Sansone, M. Vento: "Evaluating Performance of the VF Graph Matching Algorithm", Proc. of the 10th International Conference on Image Analysis and Processing, IEEE Computer Society Press, pp. 1172–1177, (1999).
26. V. Yanovski, I.A. Wagner, A.M. Bruckstein: "A distributed ant algorithm for efficiently patrolling a network", Algorithmica, 37:165–186, (2003).
27. V. Maniezzo, M. Dorigo, A. Colorni: "The ant system: Optimization by a colony of cooperating agents", IEEE Transactions on Systems, Man and Cybernatics — Part B, 26(1), pp. 29–41, (1994)
28. P. Kuner, B. Ueberreiter: "Pattern recognition by graph matching — combinatorial versus continuous optimization", Int. J. Pattern Recognition and Artif. Intell., vol. 2, no. 3, pp. 527–542, (1988).
29. J.R. Ullmann: "An Algorithm for Subgraph Isomorphism", Journal of the Association for Computing Machinery, vol. 23, pp. 31–42, (1976).
30. R. Albert, A.L. .Barabasi,: "Statistical Mechanics of Complex Networks", Reviews of Modern Physics, vol. 74, January, (2002).
31. K.A. De Jong, W.M. Spears: "Using genetic algorithms to solve NP-complete problems", in Genetic Algorithms, (J.D. Schaffer, ed.), Morgan Kaufmann, Los Altus, CA., pp. 124–132, (1989).
32. A.A. Toptsis, P.C. Nelson: "Unidirectional and Bidirectional Search Algorithms", IEEE Software, 9(2), (1992).
33. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi: "Optimization by Simulated Annealing", Science, 220, pp. 671–680, (1983).
34. Y. Rekhter, T. Li: "A Border Gateway Protocol", Request for Comments 1771, T.J Watson Research Center IBM Corporation & cisco Systems, March (1995).
35. Y. Altshuler, A.M. Bruckstein, I.A. Wagner: "Swarm Robotics for a Dynamic Cleaning Problem", in "IEEE Swarm Intelligence Symposium" (IEEE-SIS05), (2005)
36. Y. Altshuler, V. Yanovski: "Dynamic Cooperative Cleaners — Various Remarks", Technical report, (2005)
37. G. Malkin: "RIPng Protocol Applicability Statement", RFC 2081, IETF Network Working Group, January, (1997).
38. O. Gerstel, S. Zaks: "The Virtual Path Layout problem in fast networks", In Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, pp. 235–243, Los Angeles, California, August, (1994).
39. J.Y. Cai: "Lectures in Computational Complexity", Department of Computer Sciences, University of Wisconsin Madison, WI 53706 (2003).
40. S. Zaks: "Path Layout in ATM networks", Lecture Notes in Computer Science, 1338:144-177, (1997).
41. A. Felner, R. Stern, A. Ben-Yair, S. Kraus, N. Netanyahu: "PHA*: Finding the Shortest Path with A* in Unknown Physical Environments", Journal of Artificial Intelligence Research, vol. 21, pp. 631–679, (2004).
42. A. Felner, Y. Shoshani, I.A.Wagner, A.M. Bruckstein: "Large Pheromones: A Case Study with Multi-agent Physical A*", Forth International Workshop on Ant Colony Optimization and Swarm Intelligence, (2004).
43. N. Alon, J. H. Spencer: "The probabilistic method", Wiley-Interscience (John Wiley & Sons), New York, (1992) (1^{st} edition) and (2000) (2^{nd} edition).
44. A. Apostolico, Z. Galil: "Pattern Matching Algorithms", Oxford University Press, Oxford, UK.

45. V. Vazirani: “Approximation Algorithms”, Springer-Verlag, (2001).
46. S. Arora, S. Safra: “Probabilistic checking of proofs: A new characterization of NP”, Journal of the ACM, (1998).
47. M. Garey, D. Johnson: “Computers and Intractability: A Guide to the Theory of NP-Completeness”, San Francisco, CA: W. H. Freeman, (1979).