Optimizing Clustering Algorithm in Mobile Ad hoc Networks Using Genetic Algorithmic Approach

Damla Turgut
School of EECS
University of Central Florida
Orlando, FL 32816-2450
turgut@cs.ucf.edu

Sajal K. Das, Ramez Elmasri and Begumhan Turgut
Dept of Computer Science & Engineering
University of Texas at Arlington
Arlington, TX 76019-0015
{das,elmasri,bturgut}@cse.uta.edu

Abstract— **In this paper, we show how genetic algorithms can be useful in enhancing the performance of clustering algorithms in mobile ad hoc networks. In particular, we optimize our recently proposed *weighted clustering algorithm* (WCA). The problem formulation along with the parameters are mapped to individual chromosomes as input to the genetic algorithmic technique. Encoding the individual chromosomes is an essential part of the mapping process; each chromosome contains information about the clusterheads and the members thereof, as obtained from the original WCA. The genetic algorithm then uses this information to obtain the best solution (chromosome) defined by the fitness function. The proposed technique is such that each clusterhead handles the maximum possible number of mobile nodes in its cluster in order to facilitate the optimal operation of the medium access control (MAC) protocol. Consequently, it results in the minimum number of clusters and hence clusterheads. Simulation results exhibit improved performance of the *optimized* WCA than the original WCA. Moreover, the loads among clusters are more evenly balanced by a factor of ten.**

Index Terms—**ad hoc networks, clustering, genetic algorithms, performance optimization**

I. INTRODUCTION

Mobile multi-hop radio networks, also called *ad hoc* or *peer-to-peer* networks, play a critical role in places where a wired (central) backbone is neither available nor economical to build, such as law enforcement operations, battle field communications, or disaster recovery situations. This multi-cluster,mu ltihop packet radio network architecture for wireless systems should be able to dynamically adapt itself with the changing network configurations. Certain nodes, known as *clusterheads*, are responsible for the formation of *clusters*, each consisting of a number of nodes (analogous to *cells* in a cellular network), and also for the maintenance of the network topology. The set of clusterheads is known as a *dominant set*. A clusterhead does the resource allocation to all the nodes belonging to its cluster. Due to the dynamic nature of the mobile nodes, their association and dissociation to and from clusters perturb the *stability* of the network and thus reconfiguration of clusterheads is unavoidable. Thus, it is desirable to have a minimum number of clusterheads that can serve the network nodes scattered evenly in the area. An optimal selection of the clusterheads is an NP-hard problem [1], [2]. Therefore, various heuristics have been designed for this problem (e.g. [1], [5]).

In this paper, we apply *genetic algorithms* (GA) as an optimization technique to improve the performance of clusterhead election procedure. In particular, we optimize our recently proposed *weighted clustering algorithm* [4], [5]. GAs are defined as search algorithms that use the mechanics of natural selection and genetics such as reproduction, gene crossover, mutation as their problem-solving method. The goal is to be able to find out a better solution in the form of new generations that have received advantages and survival-enhancing traits from the previous generations [3], [7], [9]. An artificial-life simulation is created where survival of the fittest logic is applied for the string structures that are the living organism equivalent in real world. Even though the representation is structured, there is a randomization in data exchange to simulate the evaluation of real life forms. As each generation brings up a new set of strings by different combination of bits of pieces of the previous generation, the results are not guaranteed to come up with a generation that has a better *fitness* value but by performing different genetic operations, the probability of achieving the desired results is increased. As characteristics of an organism in encoded in a strand of DNA, genetic algorithms try to do the same in electronic genotypes that are basically just strings of bits. This bit representation can be in the form of 1's and 0's or some other form depending on the application it represents. It could so happen that binary numbers could not be sufficient enough to represent rather more complex information, behavior or characteristic. In such cases, other encoding methods are used to a string to fully and uniquely represent the data.

One of the essential factors of evolution is *mutation*. There is no guarantee that the results of a reproduction will carry traits that are fitter to survive. There could be several either predictable or unpredictable characteristics found in the new generation. In genetic algorithms, mutation is achieved by random alteration of a bit in the genotype. Given a certain rate, frequency of this alteration can be different for various applications. Another important factor in genetic algorithms is the robustness, which is defined to be the balance between efficiency and efficacy needed to survive in various environments [7]. For instance, this concept could be in the form of improving the cost efficiency of a company's product line in the same environment. These concepts are also especially of importance where one can test the efficacy of a suggested solution as many times as possible, often in the figure of hundreds of thousands of times [3]. This is why genetic algorithms are not used for testing of how well life-dependent processes work. For example, one would not use this application to make suggestions to a surgeon because the fact that in case of a bad suggestion the patient could suffer the consequences and the algorithm would learn from the training. The objective function (or the desired outcome) for a given application would be to achieve improvements to an ex-

0-7803-7632-3/02/$17.00 ©2002 IEEE

isting solution already in hand or simply finding a solution to a complex problem.

The rest of the paper will show how GA based techniques can be applied to clustering algorithms that would further enhance the performance of such algorithms. In particular, we will apply the GA technique to Weighted Clustering Algorithm (WCA) [4], [5] and demonstrate its performance improvement with respect to the number of clusters, reaffiliations, and dominant set updates. The optimized version of WCA also balances the loads among clusters which is as much as ten times better than the original WCA.

II. Optimizing a Clustering Algorithm

Let us briefly summarize the Weighted Clustering Algorithm (WCA) which selects the clusterheads based on the weight W_v of each node v. As detailed in [5], W_v is defined as

$$W_v = w_1\Delta_v + w_2 D_v + w_3 M_v + w_4 P_v$$

where Δ_v is the *degree-difference*, D_v is sum of the distances of the members of the clusterhead, M_v is the average speed of the nodes, and P_v is the accumulative time of a node being a clusterhead. The corresponding *weighing factors* are such that $\sum_{i=1}^{4} w_i = 1$. That node v with the minimum W_v is chosen to be the *clusterhead*. Once a node becomes a clusterhead, neither that node nor its members can participate further in the cluster election algorithm. The algorithm terminates once all the nodes either become a clusterhead or a member of a clusterhead. All the clusterheads are aware of their one-hop neighbors as well as the ordinary (non-clusterhead) nodes know their clusterheads. Please refer to [5] for complete details of WCA.

A. Problem Statement

We propose to optimize WCA such that the clusterheads (dominant set) is minimized while load in the network is evenly balanced among the clusters. In order to have a smaller number of clusterheads, each clusterhead must serve the maximum possible number of nodes within their clusters. By balancing the nodes among the clusters, we also assure that the lifetime of individual nodes will be increased accordingly as none of the nodes will use their processing and/or battery power more than necessary. The goal of GA is to choose the one with the lowest fitness value to be the best chromosome in that population for that generation. As Elitist model of GA is used, the index of the chromosome in the population will be saved to pass on the next generation as the genetic algorithm performs crossover, mutation and replacement.

B. GA Operations

We show how the following genetic operations are used in our approach. For more details on these operations and related concepts, refer to [7], [9].

Encoding of the data: This is also called a string representation of the given data which would be the nodes in the network under consideration. All the nodes in the search space should be present and have a unique representation. If there is a one-to-one correspondence between the search space and string representation, the design of the genetic operator would be considerably less complex. As the number of nodes can be randomly generated, it can be any number N for the given instance. These unique IDs are used to encode the chromosome using integer permutation as illustrated in Figure 2. Each chromosome will be represented as a string of integers form where each node ID is present and appears only once in the list as shown in Figure 1.

Initial Population: Since genetic algorithms can perform certain tasks in parallel, the initial population should be generated randomly. The population size is equal to what is called the pool size in genetics which is generally problem dependent, but it can also be found experimentally.

Selection: After the formation of the initial population, the fitness value for each chromosome is computed. Since the weight W_v of each node was calculated from WCA's selection procedure, GA uses those values to sum up for all the clusterheads for each chromosome. Since each chromosome has a different set of clusterheads, the total fitness value for each chromosome will be different. According to the fitness values, Roulette Wheel method is used for selection. Essentially in this method, every chromosome is assigned a percentage value that is linear to its fitness value.

Crossover: This is an essential operation despite the fact that it may eliminate the optimal solution in rare cases. The purpose is to have more diverse population. It is random in nature and dependent on the rate specified which is best suited for a given application and can be found experimentally.In this implementation, the X_Order1 method is used [10] and the crossover rate is chosen to be 0.8. In the X_Order1 method, the offspring inherits the elements between the two crossover points, inclusive, from the selected parent in the same order and position as they appeared in that parent. The remaining elements are inherited from the alternate parent in the same order as they appear in that parent, beginning with the first position following the second crossover point and skipping over all elements already present in the offspring [6].

Mutation: This operation is performed to avoid premature convergence by occasional random alternation of randomly determined bit in the given string with a specified rate. For the mutation operator, we use a swap method with mutation rate of 0.1. In this method, from the parent, we randomly select two genes at position j and i, swap them to create the new child.

Replacement: Its purpose of using an append method is to save the best strings into the next generation as it is possible to loose the best solution while the reproduction process produce a new set of solutions that replace the old (parent) solutions.

Elitism: The idea of using elitism is to update the current solution (parent string) with the new solution (child string) if and only if the new solution is better than the previous one.

Fitness value for chromosome: Compute the fitness value for child 1 and child 2 (object_func). This function is explained in Cfit Value Algorithm which computes the objective function for the fitness value.

C. Applying Genetic Algorithms

This section provides how the genetic algorithms are applied to WCA to optimize the total number of clusterheads. As can be seen in Figure 1, we have all the nodes along with their neighbor

list as well as the W_v values which are already calculated from the execution of WCA algorithm. This is stored separately in a list where each node is pointing to its neighbor(s) list as it is next position that is used to compute the object function.

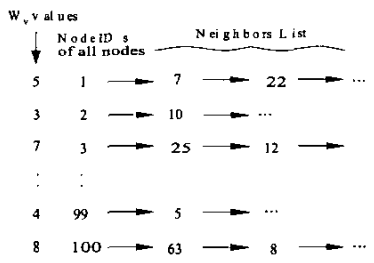

Fig. 1. WCA intermediate results

Let us propose the following genetic algorithm steps that are applied to the intermediate results of WCA.

Genetic Algorithms Steps

1) Initial Population: Randomly generate the initial population with the pool size being equal to the number of nodes in the given network. This will produce the same number of chromosomes in the form of integer strings.
2) Repeat until requirements met: While new_pool_size < old_pool_size, repeat steps 3 to 7. Repeat step 2 until the number of generation or the convergence is met.
3) Selection: Apply Roulette Wheel method with fitness values.
4) Crossover: Use X_Order1 method.
5) Mutation: Use swap method.
6) Compute objective function: Compute the fitness value of each chromosome in the population.
7) Replacement: Use append method.
8) Elitism. Check if the new children are better than the current best. If so, replace the best by the child.

Since there are certain randomly generated predefined number,say N, of mobile nodes in the network, each of which has a unique node ID in the range from 1 to N; these node IDs are used in the integer permutation to form string of integers as encoding of a single chromosome. The initial population is performed by generating the population randomly according to the pool size. It is important to note that each of these strings containing all the node IDs should not have any duplicate number; achieving completeness and uniqueness characteristics. The order in which the IDs are placed in the string should also be random and not follow a certain pattern. This is shown in Figure 2. Starting from the beginning, the algorithm goes through all the nodes from the string in the order they appear and refers back to the previous list obtained from WCA to find out the W_v values for the selection of the clusterheads.

The algorithm goes through each node in this list and checks three conditions in order to select the current node as a clusterhead. If the node under consideration is not already a clusterhead, and not a member of any of the clusterheads, and the actual number of neighbors is less than the predefined threshold value for δ (maximum allowed number of neighbors a node can have), that node is chosen to be a clusterhead and inserted into the set of clusterheads for that particular chromosome. This is

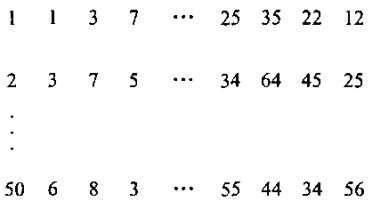

Fig. 2. Data encoded into chromosomes

shown in Figure 3. Since every node can be either a clusterhead or a member of only a single clusterhead, the selected node and its members are marked as deleted such that they would not be part of the selection procedure anymore. After going through the list once, a second run is performed for the purpose of finding out whether any node is left unassigned since every node has to be a member of a single clusterhead or a clusterhead itself. After the clusterheads are chosen, the already calculated W_v values for each node is used to find out the fitness value of the chromosome by taking the summation of W_v values of all clusterheads in this particular chromosome. It is important to note that since the order of appearance of node IDs in the encoding of the chromosomes are different, each chromosome will have a different set of clusterheads which in return will have different fitness values as computed by the following algorithm.

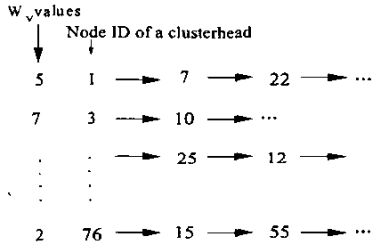

Fig. 3. Clusterhead set for a single chromosome

Cfit Value Algorithm

1) The fitness value is equal to 0 at the beginning.
2) For each gene in chromosome repeat steps 3 and 4.
3) Assign node to be equal to gene [3]. If a node is not already a clusterhead and is not already a member of another clusterhead and its node degree is less than or equal to MAX_DEGREE (constant), assign this node to be a clusterhead. Find out its W_v value (already computed from WCA). Insert its node ID to the clusterhead set. Add its W_v value to the fitness value of the chromosome it belongs to.
4) For the nodes that are leftover without any assignment, loop through the entire chromosome one more time. If a node is found that is not already a clusterhead and is not already a member of another clusterhead and its node degree is less than or equal to MAX_DEGREE, assign this node to be a clusterhead. Insert its node ID to the clusterhead set. Add its W_v value to the fitness value of the chromosome it belongs to.

We choose the append method that the new children will be

appended into the new pool. If the new children are better than the best, replace the best by the child. This method is used to prevent the solution from getting stuck at a local optima. The solution is defined to be the solution of the best chromosome of the last generation.

III. Simulation Study

We simulate a system of N nodes on a 100×100 grid. The nodes could move in all possible directions with displacement varying uniformly between 0 to a maximum value (*max_disp*), per unit time. In our simulation experiments, N was varied between 20 and 60, and the transmission range was varied between 0 and 70. The nodes moved randomly in all possible directions with a maximum displacement of 10 along each of the coordinates. Every time unit, the nodes move a distance that is uniformly distributed between 0 and *max_disp*. In the original WCA, we assumed that each clusterhead can at most handle $\delta = 10$ nodes (ideal degree) in its cluster in terms of resource allocation. Due to the importance of keeping the node degree as close to the ideal as possible, the weight w_1 associated with Δ_v was chosen high. The weights used for simulation were $w_1 = 0.7$, $w_2 = 0.2$, $w_3 = 0.05$ and $w_4 = 0.05$. Note that these values are arbitrary at this time and should be adjusted according to the system requirements. These are the same values for all weighing factors used in the original WCA. We have used `LibGA` [8] which is a library of routines written in C for developing genetic algorithms. The GA parameters are set/modified using a configuration file with no need to compile.

A. Performance Metrics

Wecomp are the performance of WCA with three performance metrics: (i) the number of clusterheads, (ii) the number of reaffiliations, and (iii) load balancing factor (LBF). The number of clusterheads in the network defines the *dominant set*. The *reaffiliation* count is incremented when a node gets dissociated from its clusterhead and becomes a member of another cluster within the current dominant set. The dominant set update takes place when a node can no longer be a neighbor of any of the existing clusterheads. These parameters are studied for varying number of nodes (N)in the system, transmission range and maximum displacement.

To quantitatively measure how well balanced the clusterheads are, we use a parameter called *load balancing factor* (LBF) as defined in [4], [5]. The load handled by a clusterhead is essentially the number of nodes supported by it. A clusterhead, apart from supporting its members with the radio resources, has also to route messages for other nodes belonging to different clusters. It is difficult to maintain a perfectly load balanced system at all times due to frequent detachment and attachment of the nodes from and to the clusterheads. As the load of a clusterhead can be represented by the cardinality of its cluster size, the variance of the cardinalities will signify the load distribution. We define the LBF as the inverse of the variance of the cardinality of the clusters. Thus,

$$LBF = \frac{n_c}{\sum_i (x_i - \mu)^2}$$

where n_c is the number of clusterheads, x_i is the cardinality of cluster i, and $\mu = \frac{N - n_c}{n_c}$ is the average number of neighbors of a clusterhead (N being the total number of nodes in the system). Clearly, a higher value of LBF signifies a better load distribution and it tends to infinity for a perfectly balanced system.

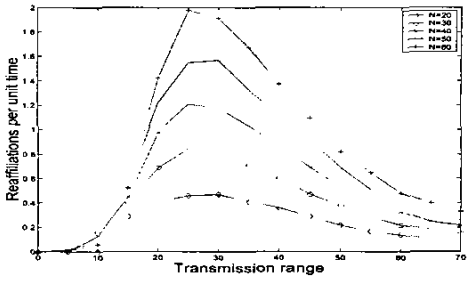

Fig. 4. Reaffiliations per unit time, *max_disp* = 5

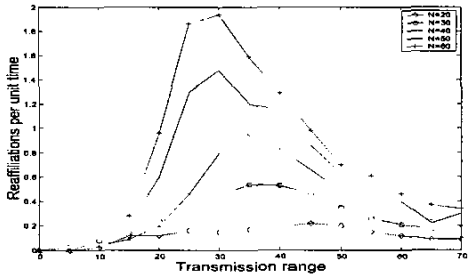

Fig. 5. Optimized WCA - Reaffiliations per unit time, *max_disp* = 5

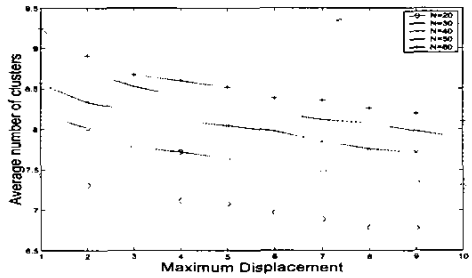

Fig. 6. Average number of clusters, *tx_range* = 30

B. Experimental Results

Figures 4 and 5 show the reaffiliations per unit time with the varying *tx_range* for original WCA and optimized WCA, respectively. For low transmission ranges, the nodes in a cluster are relatively close to the clusterhead, and a detachment is unlikely. There is an optimal transmission range for which the reaffiliations are maximum. Further increase in transmission range decreases the reaffiliations since the nodes tend to stay inside the large area covered by the clusterhead irregardless of movement of the nodes. For fewer number of nodes, the reaffiliation count is lower for optimized WCA. Figures 6 and 7 show the average number of clusterheads with the varying *max_disp* for original WCA and optimized WCA respectively. We observe that the average number of clusterheads is almost the same for different values of maximum displacement

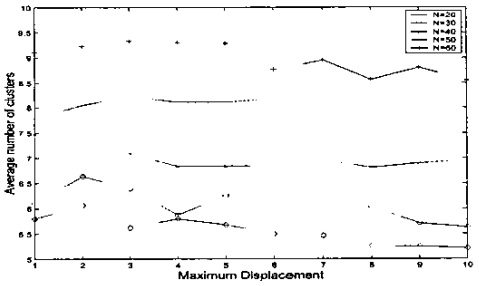

Fig. 7. Optimized WCA - Average number of clusters, *tx_range*=30

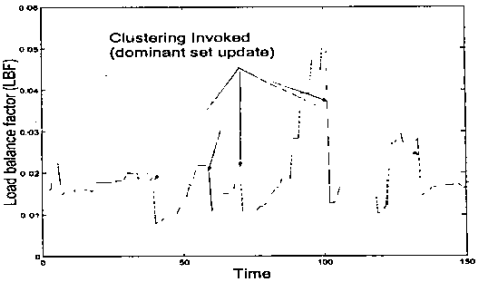

Fig. 10. Load distribution

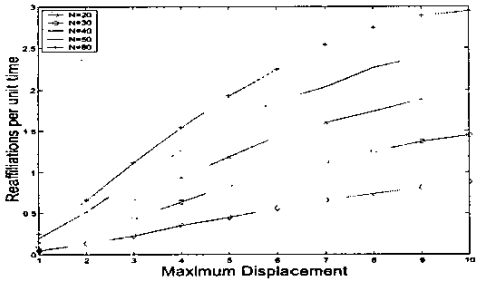

Fig. 8. Reaffiliations per unit time, *tx_range*=30

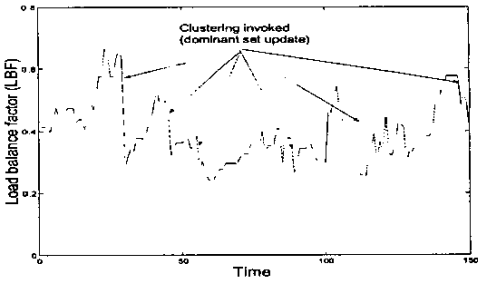

Fig. 11. Optimized WCA - Load distribution

since it simply results in a different configuration but the cluster size remains the same. We observe that the optimized WCA yields fewer number of clusters. Figures 8 and 9 show the reaffiliations per unit time with the varying *max_disp* for original WCA and optimized WCA respectively. As the displacement becomes larger, the nodes tend to move farther from their clusterhead, detaching themselves from the clusterhead and causing more reaffiliations per unit time. The reaffiliation count has considerably reduced for $N = 20$ and 30 as depicted in Figure 9. Figures 10 and 11 show how the load balancing factor (LBF) varies with time for original WCA and optimized WCA respectively. We observe that after every dominant set update, there is a *gradual* increase in the LBF. This is due to the diffusion of the nodes among clusters. While the values of LBF has varied between 0 and 0.06 in Figure 10, it went up to 0.6 in Figure 11 indicating that the GA based optimized WCA is ten times more balanced.

IV. Conclusions

In this paper, we showed how genetic algorithms can be applied to clustering techniques in mobile ad hoc networks. Weighted Clustering Algorithm (WCA) is one such algorithm which can dynamically adapt itself with the ever changing topology of ad hoc networks. We have mapped the possible solutions given by original WCA to genetic algorithm technique in order to find the better solution from a pool of solutions. Data contained in the solutions are encoded into individual chromosomes to be used in the selection process. We applied GA techniques to optimize the performance of WCA such that each clusterhead handles the maximum possible number of nodes in its cluster. The simulation results show that fewer clusterheads are obtained by applying GA to WCA than the original WCA. Also, the loads are more evenly balanced as can be seen from the improvement in the load balancing factor.

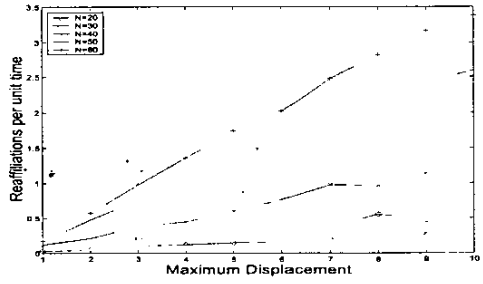

Fig. 9. Optimized WCA - Reaffiliations per unit time, *tx_range*=30

References

[1] S. Basagni, I. Chlamtac, and A. Farago, "A Generalized Clustering Algorithm for Peer-to-Peer Networks", Proceedings of Workshop on Algorithmic Aspects of Communication (satellite workshop of ICALP), Bologna, Italy, July 1997.

[2] B. Bollobas, *Random Graphs*, Academic Press, 1985.

[3] L. Chambers (Ed.), *Practical Handbook of Genetic Algorithms*, Application Volume I, CRC Press, 1995.

[4] M. Chatterjee, S.K. Das and D. Turgut, "An On-Demand Weighted Clustering Algorithm (WCA) for Ad hoc Networks", Proceedings of IEEE GLOBECOM 2000, San Francisco, November 2000, pp. 1697-1701.

[5] M. Chatterjee, S.K. Das and D. Turgut, "WCA: A Weighted Clustering Algorithm for Mobile Ad hoc Networks", Journal of Clustering Computing, (Special Issue on Mobile Ad hoc Networks), Vol. 5, No. 2, April 2002, pp. 193-204.

[6] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains", Proceedings of International Joint Conference on Artificial Intelligence, 1985.

[7] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1953.

[8] A. L. Corcoran, http://www-cgi.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/libga/0.html

[9] K.F.Ma n, K.S. Tang, and S. Kwong, *Genetic Algorithms: Concepts and Designs*,Sp ringer, 1999.

[10] *International Conference on Genetic Algorithms, Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego, July 13-16, 1991.