Constraint Programming Model for WSN Routing Problems

Course: CS 886v
Professor: Peter van Beek
Date Submitted: December 21,2007

Group Members:
1) Jun (Carol) Fung, 20269045
2) Matthew Stephan, 20098161

Table of Contents

List of Equations

List of Figures

List of Tables

Introduction

Recently, advances in wireless sensor technology have resulted in sensors being cheaper to produce and increasingly smaller in size, thus fuelling an increased interest in the use of wireless sensor networks. A sensor network consists of many sensor nodes with each sensor collecting data such as imaging, sound, temperature, and humidity and then sending that data to the data sink either directly or through other sensors [2]. Sensor networks have uses in many domains including the military, environmental, health, home, and a myriad of others [2]. A wireless sensor network is simply a sensor network comprised of wireless sensors that utilize wireless radio communication rather than wired communication. Because the sensors are immobile and the data destination, that is, the sink node, of all sensor nodes is the same, it is possible to compute the routing paths in a centralized location before activating the network and to broadcast the routing paths/decisions to each sensor. Sensors can then route the data through these pre-calculated paths. However, when computing the routing protocol for the entire wireless sensor network, there are many constraints that need to be considered such as the limited battery life for each wireless sensor and delay sensitivity of the data that is recorded by the sensors. In this report, an approach is described that utilizes constraint programming techniques for the routing problem in a wireless sensor network that allows one to find the optimal routing decisions for each node while adhering to the constraints of the wireless sensor network domain.

The problem can be described as D by D square region, in which N wireless sensor nodes are deployed randomly, following a uniform distribution. The sink node is chosen randomly inside of this square region. It is important to note that the sensing data rate on each node is constant and the energy consumption for transmitting a unit of data is a function of the distance between the two nodes that are communicating. Every node begins with the same amount of battery life remaining. As such, for each node, the transmission cost to other nodes, the receiving cost, the initial energy reserves, and the maximum length of schedule are known. The problem then becomes to find an optimal routing schedule that will maximize the network lifetime as well as take into account possible limitations on the number of routing decisions/schedules a sensor node can hold in memory. As implied earlier, this problem is important because replacing sensor nodes is expensive and sensors have limited battery and memory, therefore it is necessary to maximize the utilization of each sensor node while constraining the amount of routing decisions a sensor node can store.

This report will begin by providing background information on constraint programming and wireless sensor networks in order to facilitate an understanding of the material that will follow. Subsequently, a review of related work will be provided that includes the algorithm that the solution presented in this report is competing with. The solution will then be described in detail including the various constraints and models that are utilized as well as the variant of the arc-consistency method that was used in the solution. The report will then perform an evaluation of the solution by providing and discussing results as well as commenting on the performance. Future work will then be outlined and conclusions will be made. The code utilized for this solution is attached in the Appendix for any parties that may be interested.

Background Material

Constraint Programming

Constraint programming refers to a programming approach that places constraints that specify an acceptable solution [1]. It is modelled by specifying variables that define the problem, the domains of those variables, and the constraints that apply to the variables that comprise a desirable solution. Using this model, an acceptable or optimal solution is found by utilizing one of many approaches. The approach that is utilized in the solution presented in this report is backtracking search, which simply tries all possible combinations until an acceptable or optimal one is found [1]. However, backtracking also utilizes pruning which entails ignoring certain decisions when it is appropriate to do so.

Wireless Sensor Networks

In a wireless sensor network, sensors are deployed randomly or manually in the sensing field. Sensors take measurement of their surrounding environment regularly and send this data to another node with the purpose of that data eventually reaching the sink node through a routing path. In order to save energy, data aggregation can be performed on data to remove redundancy and to save energy [3]. As an example, as shown in Figure 1, sensor 2's sensing data is 30^{o}C, sensor 3's sensing data is 28^{o}C, and sensor 1's sensing data is 30^{o}C. Sensor 2 and sensor 3 send their data to sensor 1. If the lowest temperature is the value that the nodes are responsible for reporting, then sensor 1 takes the minimum temperature of the 2 received data packets and its local data and then sends the value from sensor 3, 28^{o}C, to its parent. In this way, node 1 only needs to send one unit of data instead of 3, which saves energy used for transmissions. The wireless sensor networks being discussed in this report are dealing with only wireless sensors networks that perform data aggregation. That is, it is assumed that all nodes will be sending out only a single packet of data.

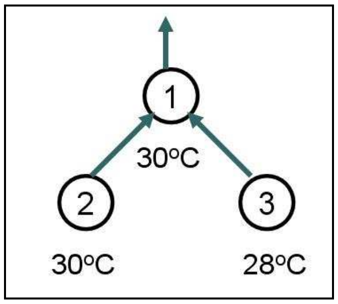


Figure 1: Example of Data Aggregation

In the recent years, much work has been done related to routing algorithms in sensor networks. Routing algorithms are typically divided into two categories, namely distributed routing and centralized routing. In a distributed routing algorithm, nodes collect information from the other nodes that they have contact with. The routing tables on nodes are dynamic and are updated based on the information they receive during message exchanging. In the centralized routing category, the routing decision is calculated by a third party offline and before the network is initialized. The routing schedule will be sent to nodes after the calculation. The routing tables on nodes are usually fixed after the initial table filling process. The routing algorithm described in this report is in the centralized category because all of the information needs to be pre calculated and sent out to all the nodes.

In the context of this report, the activity of receiving and sending packets is said to occur in a single unit of time. This is the time interval that one data packet is generated to the time that the next data packet is generated. It is assumed that the data will be sent out immediately after it is generated and it will reach the sink before the beginning of the next unit time, which is the standard case in wireless sensor networks [3]. In each unit time, the nodes should know which neighbour they send their data to. Since each node can choose only one neighbour to send their packet in each unit time, the collective routing decision of all nodes must form a spanning tree rooted at the sink node. The spanning tree for each unit of time can be different. As mentioned earlier, the transmission cost from one node to another is a function of the distance between them. The routing schedule for a single node refers to what routing decision it will make for a given unit of time. However, due to the memory limitations that sensor nodes have, the length of the routing schedule/number of different routing decisions a sensor can store must be considered in the solution proposed in this report.

Review of Previous Work

In this section, previous literature that is related to this project will be reviewed. Specifically, the routing algorithms in wireless sensor networks and related tree constraint work will be discussed.

In the distributed routing category, flooding and gossiping are two classical mechanisms. In flooding [2], each sensor broadcasts the packets to all of its neighbours and this process continues until the packet arrives at the destination node. Gossiping [2] is a slightly enhanced version of flooding where the nodes send packets to a randomly selected neighbour, which then picks another random neighbour to forward the packet to until the destination is reached. Another popular method of distributed routing is minimum cost forwarding protocol [2], which aims at finding the minimum-cost path in a sensor network. This process is comprised of two phases: the setup phase and the sending phase. In the setup phase, the cost value to transmit packets to other nodes for each node is setup. In the sending phase, the packets will be sent out by flooding and the packets and corresponding paths are disregarded when the path is not the minimum. The packets will then reach the destination through the minimum cost path. Energy aware routing [2] proposed the use of a set of sub-optimal paths instead of just one minimum-cost path. These paths are chosen by means of a probabilistic function, which depends on the energy consumption of each path. Energy-aware routing can balance the energy consumption of sensors in the network and try to avoid the situation that some nodes exhaust much faster than the others. The constraint programming algorithm proposed in this report differs in that it is a centralized routing algorithm. Furthermore, the algorithm in this report provides the optimal solution to maximize energy while the energy-aware routing solution tries only to balance the workload.

Maximum lifetime data routing [3] is a routing algorithm that belongs to the centralized routing category. In this algorithm, the cost of all possible transmission links between nodes is known beforehand and integer programming is used in conjunction with a greedy algorithm. A set of routing spanning/aggregation trees, which are trees that define how data packets are transmitted and received by the base station, are discovered and these trees' corresponding time durations are calculated. The time durations represent the amount of time that the aggregation tree will be used as the routing protocol. So, for example, a network with a lifetime of 100 rounds (of unit time) could be comprised of an aggregation tree of length 60 rounds and an aggregation tree of length 40 rounds. A notable issue with this approach is that it does not guarantee the maximum lifetime of the network. A positive implication of this, however, is that it has polynomial running time. Another problem of this approach is that the set of aggregation trees/the number of routing destinations may be too large for each of the individual sensor to store due to the sensors' limited memory capacity. As discussed earlier, the approach proposed in this report guarantees the optimal solution by using the backtracking variant of constraint programming and considers the limitation of sensor memory with regards to the schedule length. The algorithm proposed in [3] is what the constraint-model algorithm proposed in this report will be compared against.

Over the last few years graphical constraints have become of interest to the constraint programming community. In [4], two constraint models are proposed for a rooted spanning tree constraint. One uses a n-ary constraint with an inherent structure in which each node stores the index of the parent node. The domain of each node is updated to eliminate any possible cycles. The second model uses a set method and incrementally adds edges into the spanning tree set. Each time an edge is added that causes a cycle, it will be removed. [5] proposes an arc-consistency algorithm for the tree constraint which the authors claim can check in $O(|v| + |\varepsilon|)$ time. However, the problem with these approaches is that, to implement these models, it is necessary to develop code for tree pruning and domain propagation so, as such, they are very complicated to code. Rather than utilize these models for the purposes of facilitating a tree constraint, the algorithm proposed in this report introduces redundant variables representing a node's depth and utilizes its own arc-consistency algorithm in order to make the spanning tree implementation easy and fast. This will explained in more detail in a subsequent section.

Description of Solution

In this section, the solution used to solve the wireless sensor network routing problem is presented. Firstly, the decision of which existing constraint programming system to use is discussed. Following this, the constraint model will be described in detail including the transmitting and receiving model and the main constraint model. Lastly, the arc-consistency domain filtering technique that was used to select the depth of the nodes will be discussed.

Choice of Constraint Programming System

The authors of the report decided to use the Choco Constraint Programming System found at [7] as their constraint enforcement system.   The main reason the authors chose this solution is that it provided a Java library for the constraint satisfaction domain and both of the authors are very comfortable with Java programming. Furthermore, Choco supports many different advanced constraints such as the all different and global cardinality constraints [7]. It should also be noted that the system has an in-depth Javadoc and User Guide that assisted the authors immensely. Lastly the system is continually updated and supported as it includes algorithm improvements such as newer generalized arc consistency and arc consistency algorithms [7].

Constraint Model

Transmitting and Receiving model

After the deployment of sensor nodes in a wireless sensor network, the location information of nodes can be detected by a Global Position System (GPS) and then sent to the control center. At this point, the distance between any pair of nodes will be known. The algorithm discussed in this report uses the formulae presented in [3] to determine transmission cost and receiving cost for each sensor node. As shown in Equation 1, the communication cost between any two nodes is a function of the distance between two nodes and the size of the packet. In Equation 1, $TX_{i,j}$ denotes the energy cost on node i when node i transmits one data packet to node j. For Equation 2, RX_i denotes the energy cost for node i to receive a packet. For both Equation 1 and Equation 2, ε_{elec} is the energy required to run the transmitter or receiver circuitry, respectively, ε_{amp} is the energy required for transmitter amplifier, and k is the number of bits in the packet. In the constraint model described in this report, it is important to note that nodes are deployed in the deployment region randomly via uniform distribution and that the transmission cost matrix is based on the nodes' geological relations.

Equation 1: Calculation of transmission cost [3]

$$\boxed{TX_{i,j} = \varepsilon_{elec} \times k + \varepsilon_{amp} \times d_{i,j}^2 \times k}$$

Equation 2: Calculation of Reception Cost [3]

$$\boxed{RX_i = \varepsilon_{elec} \times k}$$

Constraint model

In this section the constraint model is presented with respect to its variables, constraints, and optimization goal.

Variables

The first variable used is the parent index $Pnode_{ik}$, which is the index of the parent node for node i in round k. It is important to note that the term "round" refers to the period of time in which a particular routing/aggregation tree will be used. $Pnode_{0k} = 0$ for all k because node 0 is the sink node and it sends nowhere/ to itself. For all other nodes $i \neq 0$, $Pnode_{ik} = j$, where $i \neq j$. That is, as shown in Equation 3, the domain of the parent node variable for all the sensor nodes is 0 to n-1 excluding node i itself.

Equation 3: Domain of Parent Node for Sensor Nodes

$$\boxed{dom(Pnode_{i\neq 0}) = \{0,1,...n-1\} \setminus \{i\}}$$

The next variable used is Depth D_{ik}, which is the depth of node i in the tree in round k. $D_{0k} = 0$ because the sink node is the root of the tree. In the case of sensor nodes or i not equal to 0, then $D_{ik} = j$ means the depth of node i is j, where the domain, as shown in

Equation 4, is 1 to n-1 because depth of the sensor nodes could be anywhere from right below the sink to n-1 in the case of a straight-line tree. The purpose of introducing the depth variable is to prevent cycles in the spanning tree. It is important to note that the introduction of D_{ik} does not produce redundant solutions for $Pnode_{ik}$. This can be proved via proof by contradiction. Assuming this statement is not true, then there exists two solution sets $Pnode + D$ and $Pnode' + D'$, where $Pnode = Pnode'$ and $D \neq D'$. However, this cannot be true because every unique spanning tree can have only unique assignments of depth for each node. Therefore, a contradiction exists and the negation of our statement must be false.

Equation 4: Domain of Depth for Sensor Nodes

$$\boxed{dom(D_{i\neq 0}) = \{1,2,...,n-1\}}$$

The last variable utilized is round duration L_k, which is the amount of unit time that the spanning tree utilized in round k is to be used as the routing tree/protocol. In all cases, $k \leq M$ where M is the maximum length of the schedule and is specified by the user based upon a sensor's specific memory capabilities. That is, M would be set to the maximum amount of routing/spanning trees that an individual sensor within a wireless sensor network can store. The domain of L_k, as shown in Equation 5, is simply 0 in the case of a redundant round to the upper bound for that round duration. The upper bound for a round duration can be calculated by finding the maximum lifetime of the spanning tree contained in that round. The maximum lifetime is the minimum value found by investigating all the nodes and finding the energy reserve divided by the amount of energy used each unit time.

Equation 5: Domain of Round Duration Variable

$$\boxed{dom(L_k) = \{0,...,Upper(L)\}}$$

Spanning Tree Constraints

The constraints necessary to ensure only spanning trees are considered include the connectivity constraint and the no-cycle constraint. The connectivity constraint enforces that all the nodes are connected with only one other node. This is implied in the constraint model because, as stated in Equation 3, all nodes except the sink node have one parent node and the parent node cannot be themselves. The no-cycle constraint ensures that there cannot be any cycle in the resulting tree. This can be enforced by the constraint on the depth variables, as indicated in Equation 6, which says that the depth of a child node is always one level more than the parent node for each round, *k*.

Equation 6: No-Cycle Constraint on Depth Variable

$$\boxed{D_{ik} = D_{Pnode(i,k)} + 1}$$

Energy constraints

As noted earlier, each sensor node consumes energy when transmitting or receiving a data packet. The total energy reserve of each node is given by ER_i and is something that would be provided by the user and stored in the form of an n by 1 matrix. Given this, the energy constraint becomes what is shown in Equation 7 below, which is only a slight variation to the energy constraint found in [3]. This constraint is saying that for each node in the wireless sensor network, the total energy reserve for each node must be greater than or equal to the summation of the total energy consumed by both transmitting, $TX[i][Pnode_{ik}]$, and receiving, $RX[i]\times \sum_{Pnode_{jk}=i} 1$, data packets for all the rounds, $\sum_{k-1}^{M} L_k$.

Equation 7: Energy Constraint

$$\boxed{\begin{array}{l} \forall i \varepsilon \{0,1,...,n-1\}, \\ ER_i \geq \sum_{k=1}^{M} L_k \times (TX[i][Pnode_{ik}] + RX[i] \times \sum_{Pnode_{jk}=i} 1) \end{array}}$$

Optimization Goal

The last constraint considered is the one providing the optimal solution based on all of the spanning trees discovered using the constraints described previously. Since the optimization goal is to find the maximum lifetime of the network, it is necessary only to maximize the sum of durations of all rounds because these are equivalent, which is shown in Equation 8. Optimality is accomplished through the use of backtracking search, which guarantees the optimal solution if allowed to run to its entirety.

Equation 8: Maximization of Round Durations Constraint

$$\max(\sum_{k=1}^{M} L_k)$$

Domain Filtering Through Arc Consistency

The arc-consistency domain filtering will be executed for the assignment of the depth variable described in the constraint model just presented. Whenever a new parent index is assigned to a node, the domain of depth will be filtered to satisfy the constraint described in Equation 6. An example of this domain-filtering arc-consistency technique is illustrated in Figure 2. In the beginning, T1, all source nodes have a full domain of {1,..,4}. At time period T2, node 1 is been assigned a parent node 0, therefore, its depth can be only 1 according to the constraint in Equation 6. At time T3, node 2 is assigned a parent node 3. At this point, arc consistency is performed and the depth variable domains for node 2 and node 3 are filtered accordingly. At T4, node 4 becomes the parent node of node 3. Arc consistency is performed once again and the depth domains for node 2,3, and 4 are {3,4}, {2,3}, and {1,2}, respectively. At this point, a parent node for node 4 is searched for and, according to the constraint in Equation 6, the parent node of node 4 must have a depth of one level smaller than node 4, implying a value of 0 or 1. At time point T5, the assignment of node 2 as the parent of node 4 is illegal since it violates the depth constraint, thus, the satisfied node can be only node 0 or node 1. In T6, node 0 is assigned to be the parent node of node 4 and a rooted spanning tree is yielded.

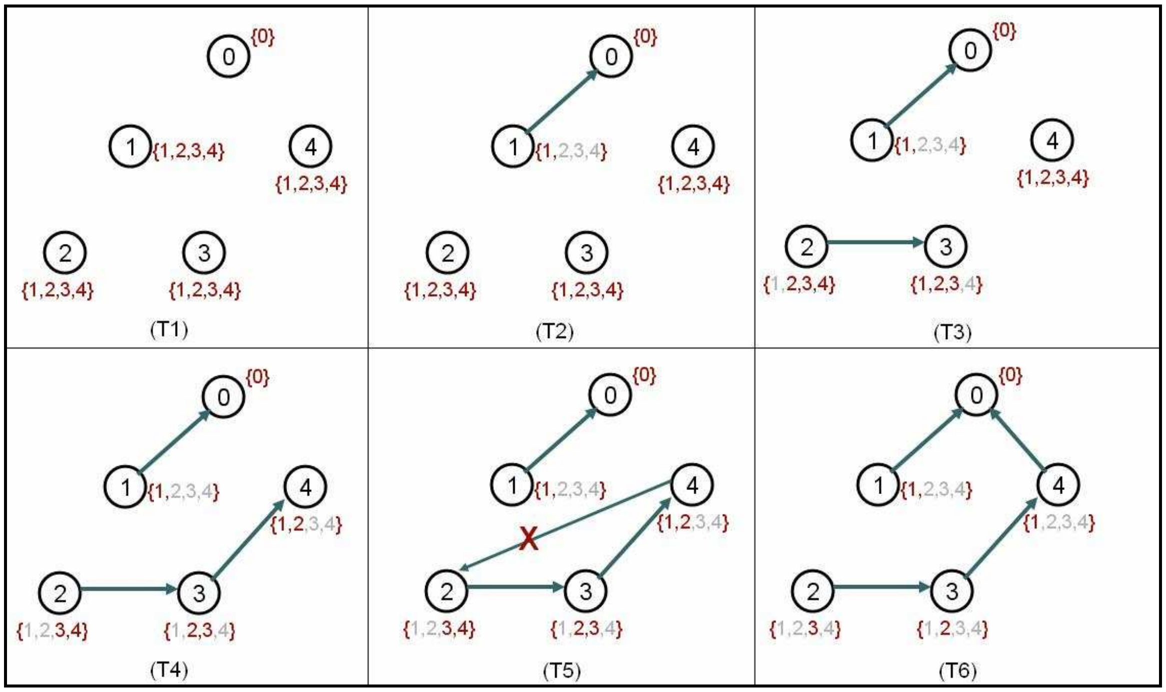


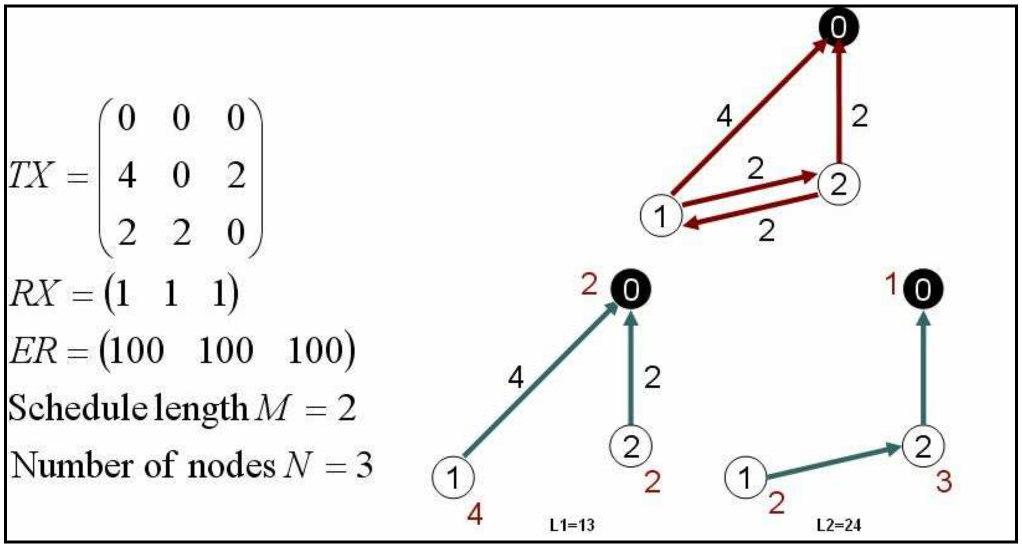
Figure 2 A illustration of arc consistency propagation in spanning tree search

In this procedure, arc consistency is performed to filter the domain of the nodes' depth. The complexity to check the spanning tree constraint is $O(|\varepsilon|)$, where, $|\varepsilon|$ is the number of edges in the graph. This result is competitive with the tree arc consistency complexity in [5], which was $O(|v| + |\varepsilon|)$. The fact that it is competitive and easier to implement motivated the decision to use this simplified arc consistency method compared to the one presented in [5].

Evaluation

Results

In this section, the preliminary results of the constraint programming algorithm for wireless sensor routing are provided. Figure 3 shows the sample results for a small network of 3 nodes. In this example, a directed graph, shown with red arrows, is given on the top showing the transmission cost between any pair of nodes. The transmission cost matrix, receiving cost array, energy reserve array are given on the left of the figure. In this case, the receiving cost was set to the value one and the energy reserve for each sensor, including the sink, was set to 100. Also, the maximum schedule length, M, is set to two which means that only two different spanning trees are allowed in the solution. The results show that the optimal network lifetime was found by utilizing the spanning trees underneath left and underneath right of the top graph, tree 1 and tree 2, respectively. Specifically, tree 1 will run for 13 unit time and tree 2 will be executed for 24, as indicated by the values of L_1 and L_2, respectively. This implies that the maximum lifetime of the network is 37 cycles.



$TX = \begin{pmatrix} 0 & 0 & 0 \\ 4 & 0 & 2 \\ 2 & 2 & 0 \end{pmatrix}$
$RX = (1 \quad 1 \quad 1)$
$ER = (100 \quad 100 \quad 100)$
Schedule length $M = 2$
Number of nodes $N = 3$
L1=13
L2=24

Figure 3: Result for a Small Wireless Sensor Network

In another set of experiments, nodes are added randomly, via a uniform distribution, into a 20x20 region. The transmitting cost/matrix is then calculated as a function of the square distance between the nodes. The receiving cost and battery budget remain the same as shown in Figure 3. Table 1 shows that the network lifetime varies with the number of nodes deployed in the region and the maximum schedule length M. As expected, the network lifetime increases with the number of nodes in the region. This is because the increase of nodes increases the density of the network, which, in turn, decreases the average distances among nodes. As such, the average cost to transmit packets is lower. The increment of schedule length/number of allowable spanning trees also increases the network lifetime. This is because using multiple spanning trees can decrease the heavy workload put on individual nodes and can allow nodes to take turns being the high-energy consumer/busy worker. Therefore, the network lifetime increases. As shown by these results, the constraint model algorithm provides the optimal solution and allows the sensors' memory capacity to be considered.

Table 1: Network Lifetime given Network Size and Schedule Length

	3 Nodes	4 Nodes	5 Nodes	6 Nodes	7 Nodes	8 Nodes
1 Tree Allowed M=1	12	12	12	12	12	16
2 Trees Allowed M=2	12	13	13	14	14	16

Regarding the question of whether or not the test cases/results shown are representative, the answer is that they are for small-scale wireless sensor networks only. As will be addressed in the performance section, it is only possible to run the algorithm on small-scale networks with less than 15 nodes. The authors of this report, however, are quite confident that if the performance issues are addressed that the algorithm will work for medium and large scale networks because the constraint model appears mathematically and logically.

Performance

The programming running time is graphed in Figure 4. In this experiment, only one spanning tree is allowed, that is, M is set to 1. As shown in the graph, the running time of the program increases slowly in the beginning in conjunction with the number of nodes. As soon as the number of nodes reaches 15, however, the running time jumps up dramatically. This is a typical exponential running time increment. The justification for this is that the number of spanning trees in the n-nodes network increases in the order of $O(n^{n-2})$ as noted in Cayley's formula [6]. Although constraint programming can improve the performance of a NP-Complete search problem greatly in many cases, the worst case running time is still high.

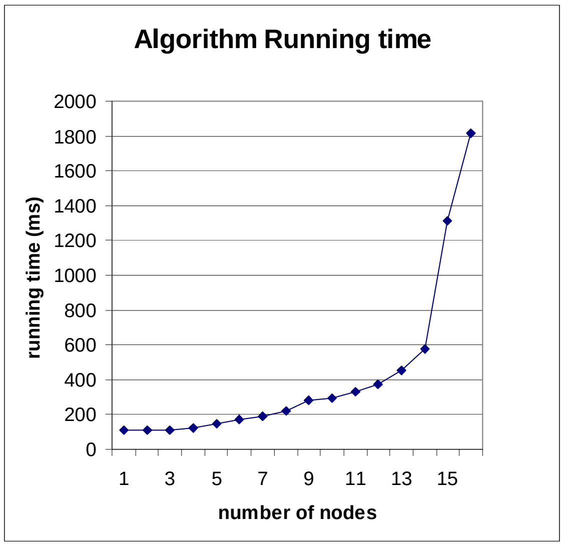


Figure 4: Program Running Time versus Number of Nodes

Compared to the integer programming and greedy algorithm model provided in [3], the constraint programming model running time is not satisfactory. Currently, it can only be used on small networks while [3] provides a polynomial running time model and can be used on median scale network. It is important to note, however, that the constraint model algorithm has in no way attempted to be optimized and more work can be done to refine the model and improve the running efficiency of the model, as will be addressed when discussing future work.

Future Work and Conclusions

Firstly, the authors of this report are going to investigate variable-ordering and value-ordering heuristics, as discussed in [1] to see if the algorithm can be improved. Typically, constraint programming solutions are far from optimized when first developed and heuristics such as these often yield great performance benefits. It would be ideal if the running time could be improved to a point where medium and possibly large-scale sensor networks can have an optimal solution that is yielded from the algorithm presented in this paper.

Another area that the authors want to explore is to attempt to improve the algorithm running time by using local search. While utilizing local search instead of backtracking while lose the solution's guarantee of an optimal solution, the running time would likely be less than or equal to the solution presented in [3], thus providing a constraint programming alternative to the solution in [3] for medium and large scale networks.

Another possible area of work is to improve the solutions from [3] by applying local search on their resulting spanning trees. That is, execute the algorithm as presented in 2, but then, rather than just using the resulting spanning trees, utilize local search to find better solutions. Lastly, it would be interesting to investigate a hybrid approach that uses Constraint Programming with Integer Programming.

The goal of any wireless sensor network routing protocol should be to maximize the lifetime of the network due to the expense incurred in replacing nodes within the network. The algorithm presented in this report provides an optimal solution of network lifetime for small-scale wireless sensor networks and allows the memory of the sensor nodes to be considered in the calculations. This is accomplished by utilizing constraint programming with constraints that ensure that the routing schedule forms a spanning tree and that network lifetime is extended as much as possible. The strength of the algorithm lies in its optimality guarantee, but the weakness that comes as a result of this is that the running time grows exponentially with the number of nodes. The authors are quite hopeful, however, that the runtime can be improved significantly in order to allow this algorithm to be usable on both medium and large-scale wireless sensor networks. If this was the case, optimal solutions would be achievable for all wireless sensor networks and the benefits would be considerable.

References

[1] F. Rossi, P. van Beek, T. Walsh (eds.) Handbook of Constraint Programming. Elsevier. 2006.

[2] K. Akkaya, M. Younis, A survey on routing protocols for wireless sensor networks, Journal of Ad Hoc Networks, 2003.

[3] K. Kalpakis, K. Dasgupta, P. Namjoshi, Maximum lifetime data gathering and aggregation in wireless sensor networks, in: Proceedings of IEEE International Conference on Networking (NETWORKS '02), Atlanta, GA, August 2002.

[4] P. Prosser and C. Unsworth, Rooted tree and spanning tree constraints, in 17th ECAI Workshop on Modeling and Solving, 2006

[5]N. Beldiceanu, P. Flener, and X. Lorca. The tree constraint. In: R. Bartak and M. Milano (editors), Proceedings of CP-AI-OR'05, pp. 64-78. Lecture Notes in Computer Science, volume 3524. © Springer-Verlag, 2005

[6]A. Cayley, A theorem on trees, Quart. J. Math. 23 (1889), 376-378; Collected Papers, Cambridge f 13 (1897), 26-28.

[7] "Choco constraint programming system", http://choco-solver.net/ (last accessed December 19, 2007)

Appendix

Routing Sensor Code with Choco Constraints

```
// **************************************************
// *    CS886 Course Project                    *
// *    University of Waterloo (2007)             *
// *    Sensor network routing problem            *
// *    using constraint programming model        *
// *    problem statement:                        *
// *    There exist a set of sensor nodes and      *
// *    a set of edges. The weight of the edge(ij) *
// *    represents the energy cost to transmit one *
// *    packet from sensor i to sensor j. We also  *
// *    know the energy cost to receive one packet *
// *    for each sensor. The budget of the sensors *
// *    are give and the data can be aggregeated in*
// *    each sensor. We need to find a packet      *
// *    delivering schedule for each sensor so that*
// *    the life time of the whole network is max  *
// *                                            *
// *    authors:Jun Fung, Matthew Stephan          *
// *                                            *
// **************************************************

import choco.*;
import choco.real.*;
import choco.set.SetVar;
import choco.integer.IntDomainVar;
import choco.integer.IntExp;
import choco.integer.IntVar;
import choco.set.search.*;
import java.util.*;
import choco.real.exp.RealIntervalConstant;

public class RoutingSensor_schedule {

  public static int N = 3;//number of sensors in the network
  public static int M = 2;//maximum number of schedule round can fit in the sensor memory

  public static int[][] tmp = {{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{10,0,8,7,12,3,8,8,11,13,6,13,13,14,4,9,5,9,8,10},
{7,8,0,10,5,5,5,5,15,16,2,11,5,6,10,7,11,5,2,8},
{7,7,10,0,15,8,13,6,5,6,8,7,15,14,11,5,3,7,11,5},
{12,12,5,15,0,9,5,10,21,21,7,16,1,6,13,13,17,10,4,13},
{9,3,5,8,9,0,5,7,13,15,3,13,10,11,5,9,8,8,5,10},
{12,8,5,13,5,5,0,10,18,19,5,16,6,9,8,12,13,10,3,13},
{2,8,5,6,10,7,10,0,11,11,5,6,10,8,12,2,9,1,7,3},
{11,11,15,5,21,13,18,11,0,2,14,10,21,19,15,9,6,12,16,9},
{11,13,16,6,21,15,19,11,2,0,15,9,21,19,17,9,8,12,17,8},
{7,6,2,8,7,3,5,5,14,15,0,11,7,8,8,7,9,5,2,8},
{4,13,11,7,16,13,16,6,10,9,11,0,15,12,17,4,11,6,13,3},
{11,13,5,15,1,10,6,10,21,21,7,15,0,5,14,12,17,10,5,13},
{8,14,6,14,6,11,9,8,19,19,8,12,5,0,16,10,17,7,7,10},
{14,4,10,11,13,5,8,12,15,17,8,17,14,16,0,13,9,12,9,14},
{2,9,7,5,13,9,12,2,9,9,7,4,12,10,13,0,8,2,9,1},
{10,5,11,3,17,8,13,9,6,8,9,11,17,17,9,8,0,10,12,8},
{2,9,5,7,10,8,10,1,12,12,5,6,10,7,12,2,10,0,7,3},
{9,8,2,11,4,5,3,7,16,17,2,13,5,7,9,9,12,7,0,10},
{2,10,8,5,13,10,13,3,9,8,8,3,13,10,14,1,8,3,10,0}};
  public static int[][] TX = new int[N][N];//transmition cost per unit of data
  public static int[] RX = new int[N];//receiving cost per unit of data
  public static int[] ER = new int[N];//energy budget for each sensor

  public static void main(String[] args) {
        Date start = new Date();
        System.out.println(new RoutingSensor_schedule().demo());
        System.out.println("Running time = "+(new Date().getTime()- start.getTime()));
}

public String demo() {

        for (int i=0; i<N; i++){
                    RX[i]=1;
                    ER[i]=100;
                    for (int j=0; j<N; j++)
                          TX[i][j]=tmp[i][j];
        }
  Problem pb = new Problem();

  // create variables
  IntDomainVar[][] OE = new IntDomainVar[M][N];//out edge array

  //Rank of nodes, parents nodes have higher rank that children nodes. We use this to prevent cycles.
  IntDomainVar[][] Rank = new IntDomainVar[M][N];
  IntDomainVar[][] RankPlus = new IntDomainVar[M][N];//Rank + 1

  //connection matrix conn[k][i][j]=1 represents sensor i send data to sensor j at round k
  IntDomainVar[][][] Conn = new IntDomainVar[M][N][N];
  IntDomainVar[][][] Conn_rev = new IntDomainVar[M][N][N];//connection matrix reverse

  IntDomainVar[][] Cost = new IntDomainVar[M][N];//The communication cost for sensors per unit time in each round
  IntDomainVar[] roundLen = new IntDomainVar[M];//The time span for each round
  IntDomainVar life = pb.makeBoundIntVar("lifetime",0,ER[0]);//the lifetime of the network(min lifetime of sensors)
  IntDomainVar[][] Sensor_erg = new IntDomainVar[N][M];//the communication cost for sensors in each round


        //instant the variables and append them to the problem
        for (int k = 0; k<M; k++)
              for (int i = 0; i<N; i++){
                    OE[k][i] = pb.makeEnumIntVar("OE[" + k + "][" + i + "]", 0, N-1);
                    Rank[k][i] = pb.makeEnumIntVar("Rank[" + k + "][" + i + "]", 0, N-1);
                    RankPlus[k][i] = pb.makeEnumIntVar("RankPlus[" + k + "][" + i + "]", 1, N);
                    Cost[k][i] = pb.makeBoundIntVar("Cost[" + k + "][" + i + "]", 0, ER[i]);
                    Sensor_erg[i][k] = pb.makeBoundIntVar("Sensor[" + i + "][" + k + "]", 0, ER[i]);
              }
        for (int k = 0; k<M; k++)
              for (int i=0; i<N; i++)
                    for (int j=0; j<N; j++){
                          Conn[k][i][j] = pb.makeEnumIntVar("Conn[" + k + "][" + i + "][" + j + "]", 0, 1);
                          Conn_rev[k][i][j] = pb.makeEnumIntVar("Conn_rev[" + k + "][" + i + "][" + j + "]", 0,
1);
                    }
        for (int k = 0; k<M; k++)
              roundLen[k] = pb.makeEnumIntVar("L[" + k + "]", 0, ER[0]); //upper bound ??


  // add constraints to problem
        //constraints among variables (relation between IE and OE)
        for (int k = 0; k<M; k++){
              pb.post(pb.eq(OE[k][0],0)); //OE[k][0] = 0 sink has no out edge
              pb.post(pb.eq(Rank[k][0],N-1) );//sink has the highest rank
        }
        for (int k=0; k<M; k++)
              for (int i=0; i<N; i++){
                    pb.post(pb.eq(Conn[k][0][i],0));//sink node has no out edge
              }
        for (int k=0; k<M; k++)
              for (int i=1; i<N; i++){
                    pb.post(pb.eq(pb.sum(Conn[k][i]),1));//only one out edge is allowed for each node
              }

        IntVar one = pb.makeConstantIntVar(1);//temp usage

        for (int k = 0; k<M; k++)
```

```
                        for (int i = 1; i<N; i++){
                                pb.post(pb.neq(OE[k][i],i)); //edge should point to some other nodes
                                pb.post(pb.nth(OE[k][i],Rank[k],RankPlus[k][i]));//parent node rank = child node rank+1
                                pb.post(pb.nth(OE[k][i],Conn[k][i],one));//the parent node always has one level higher rank
                        }
                for (int k = 0; k<M; k++)
                        for (int i = 0; i<N; i++)
                                pb.post(pb.eq(RankPlus[k][i],pb.plus(Rank[k][i],1) ));

                for (int k = 0; k<M; k++)
                        for (int i=0; i<N; i++)
                                for (int j=0; j<N; j++)
                                        pb.post(pb.eq(Conn[k][i][j],Conn_rev[k][j][i]));

                //energy constraints for each sensor
                for (int k = 0; k<M; k++)
                        for (int i=0; i<N; i++)
                                pb.post(pb.eq(Cost[k][i],pb.plus(pb.sum(Conn_rev[k][i]),pb.scalar(TX[i],Conn[k][i])) ));

                for (int i = 0; i<N; i++){
                        for (int k = 0; k<M; k++)
                                pb.post(pb.times(Cost[k][i],roundLen[k],Sensor_erg[i][k]));
                        pb.post(pb.leq(pb.sum(Sensor_erg[i]),ER[i]));
                }
                pb.post(pb.eq(pb.sum(roundLen),life));

                //Solver.setVerbosity(Solver.PROPAGATION);


    Solver solver = pb.getSolver();
    //pb.solve(false);
    pb.maximize(life,false);

    System.out.println("feasible: " + pb.isFeasible());
    System.out.println("nbSol: " + pb.getSolver().getNbSolutions());

    //packSolutions(pb);

    StringBuffer s = new StringBuffer();

    s.append("The " + solver.getSearchSolver().solutions.size() + " last solutions (among " +
        solver.getNbSolutions() + " solutions) are:\n");
    List solutions = solver.getSearchSolver().solutions;
    s.append("The solutions for rooted spanning tree are: \n \n");
    s.append(solutions.size() + " solutions : \n");
    for (int i = 0; i < solutions.size(); i++) {
      Solution solution = (Solution) solutions.get(i);
      s.append("lifetime:"+solution.getValue(0) + " ");
                 for (int j = 0; j<M; j++){
                for (int k = 1; k<5*N; k=k+5){
                        s.append(solution.getValue(j*N*5+k) + " ");
                        }
                        s.append("(L["+j+"]=" +solution.getValue(M*N*5+M*N*N*2+j+1) + ") ");
                 }
      s.append("\n");
    }
    s.append(pb.solutionToString());
    return s.toString();
  }

}
```

Code use to Generate Sensors and Calculate Transmission Cost

```
// **************************************************
// *    CS886 Course Project                     *
// *    University of Waterloo (2007)             *
// *    Generate sensorsin a square region,        *
// *    calculate the communication cost           *
// *    for each pair of node and store the        *
// *    into a transmission matrix.              *
// *                                     *
// *    authors:Jun Fung, Matthew Stephan          *
// *                                     *
// **************************************************
import java.util.*;
public class GenMatrix{
		public static int N = 10;
		public static void main(String[] args) {
			int[] x_axis = new int[N];
			int[] y_axis = new int[N];
			int[][] trans = new int[N][N];
			Random r = new Random(5);

			//generate positions
			for (int i=0; i<N; i++)
			{
				x_axis[i] = r.nextInt(N);
				y_axis[i] = r.nextInt(N);
			}
			//calculate transmission cost
			for (int i=0; i<N; i++)
				for (int j=0; j<N; j++)
				{
					if(i==0)
						trans[i][j] = 0;
					else if(i==j)
						trans[i][j] = 0;
					else{
						trans[i][j]  = (int)Math.pow((x_axis[i]-x_axis[j])*(x_axis[i]-x_axis[j]) +
(y_axis[i]-y_axis[j])*(y_axis[i]-y_axis[j]),0.5);
					}

				}
			//print the cost
			for (int i=0; i<N; i++){
				System.out.print("{");
				for (int j=0; j<N; j++){
					System.out.print(trans[i][j]);
					if(j<N-1) System.out.print(",");
				}
				System.out.println("},");
			}

			boolean valid = true;
			for (int i=1; i<N; i++)
				for (int j=0; j<N; j++)
					if (trans[i][j]==0 && (i!=j))
						valid=false;
			System.out.println("valid data:"+valid);

		}
}
```