Energy-Efficient Task-Mapping for Data-Driven Sensor Network Macroprogramming Using Constraint Programming

Farshid Hassani Bijarbooneh, Pierre Flener, Edith Ngai, and Justin Pearson

Department of Information Technology
Uppsala University, Box 337, SE – 751 05 Uppsala, Sweden
{farshid.hassani, pierre.flener, edith.ngai, justin.pearson}@it.uu.se

Abstract Using constraint programming (CP), we address the task-mapping problem in data-driven macroprogramming for wireless sensor networks (WSNs). A task graph representing the flow of data among tasks assists the application developer in specifying the features of a WSN at a high level of abstraction. A problem that arises in this context is how to map the tasks to nodes in the target network before the deployment of sensors, in order to achieve an energy-efficient WSN. This problem is slightly different from the deployment problem for distributed systems. We take a published formulation of the WSN task-mapping problem solved by mixed integer programming (MIP) solvers, and rewrite it much more naturally as a constraint program, using off-the-shelf CP components. On realistic instances of real-world applications of the problem, we show that our CP model results in significantly better runtimes than the MIP model.

1 Introduction

Wireless sensor networks (WSNs) operate as distributed systems where sensors cooperatively monitor or control conditions in an environment, such as temperature, speed, or pressure [1]. A node in a WSN consists of at least a sensor, processor, radio transmitter, and a battery. It is of great concern to reduce the energy consumption at each node before deploying the network to the environment. The lifetime of a WSN depends critically on the energy consumption of the nodes, especially in cases where the battery cannot be charged once it is drained [4].

In a WSN, a *node* is assigned to perform tasks. *Tasks* are pieces of code implementing applications of the network. The entire network repeats the same behaviour over a time period called a *round*. Nodes have an initial energy level, which drops as they communicate and process the assigned tasks in each round of the network [9].

Tasks are grouped into three categories: sensing tasks, operative tasks, and actuator tasks. A *sensing task* calls a sensor to collect data. For example, invoking a sensor to measure temperature in a room is performed by a sensing task.

An *operative task* performs operations on data that has been gathered by the sensing tasks. For example, taking the temperature from differently positioned sensors in a room and computing the average is performed by an operative task. Finally, an *actuator task* performs an action to affect the environment, which is based on data processed by operative tasks. For example, consider a task that turns on a heater to warm a room. Sensing tasks and actuator tasks can only run on nodes with the appropriate sensors or actuators, while operative tasks are free to run on any node with sufficient computational resources.

Programming a task for each sensor individually is a very time consuming process. Here we use a methodology that allows defining and deploying tasks regardless of WSN architecture. In the WSN context, *data-driven macroprogramming* refers to an approach that facilitates sensor network programming by specifying the features of a WSN as a task graph [2,6]. Using this method, a task graph is created based on data flow that is independent of the network topology.

Data-driven macroprogramming poses many challenges such as how to map the task graph efficiently onto the nodes in a WSN. A task mapping can be made more efficient by reducing the energy consumption. Note that task mapping is not only initiated at the deployment of the macroprogram, but also at any time during the operation of the WSN if the energy level of a node drops under a certain level.

In this paper, we optimise the task-mapping process in a multi-hop [1] heterogeneous [8] WSN to achieve overall minimum energy consumption by the sensors. A *multi-hop* WSN allows several nodes to be on the path between two nodes, so nodes have to consider routing information. A *heterogeneous* WSN is a network that is able to provide several wireless services simultaneously.

A data driven task graph is represented as a directed acyclic graph, which is used as input to the task mapping process. During task mapping, we consider task computation costs; data firing rates and sizes; node routing costs; and placement constraints. The task graph and the general modelling framework are the same as those in [2,13]. We take their framework and instance data related to two realistic instances of real-world applications of the problem: the monitoring of heating, ventilation, and air conditioning [3] and highway traffic management [7].

We *contribute* to the WSN task mapping problem (which differs from the deployment problem for distributed systems) by implementing our model as a constraint program and showing that it achieves at least an order of magnitude speedup compared to the mixed integer programming (MIP) model of [13] for realistic instances of real-world applications.

The rest of this paper is organised as follows. In Section 2, we introduce the task mapping problem in more detail, as well as the two real-world applications of the problem. In Section 3, we explain a mathematical model of the task mapping problem. We summarise the MIP implementation of this model and introduce our CP implementation thereof, using a standard distinguishing feature of CP, namely the ability of indexing a matrix by decision variables, thereby referring to an unknown element of the matrix, using the *element* constraint [15]. In Section 4, we report on the experiments we made on several realistic instances

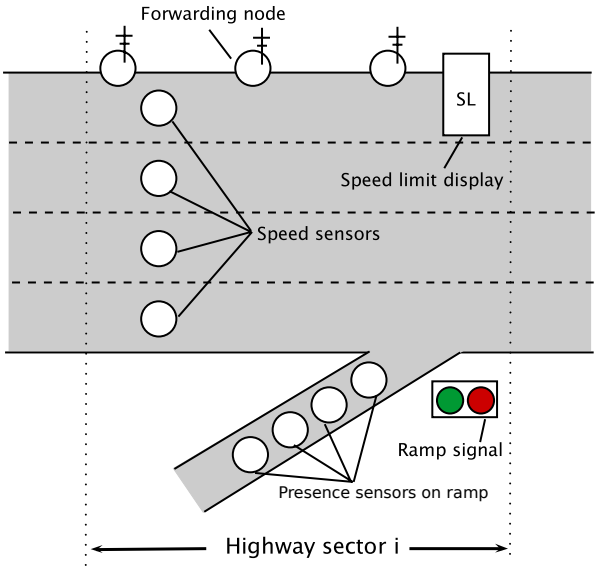

Figure 1. Node placement in the highway traffic management system

for the two applications of the problem. Finally, in Section 5, we conclude, discuss related work, and outline future work.

2 The Problem and Some Applications

In a task mapping problem, we need to consider the cost of routing a message between any two nodes via any particular node, which means that on a WSN with n nodes we will have a cost routing table of size n^3. The combinatorial problem is to map optimally the tasks with regard to such a three-dimensional cost routing matrix, considering the cost of sensing, operative, and actuator tasks. A current method for optimal task mapping involves modelling and solving it with a MIP solver [13], which shows poor performance on large-size instances of the problem (up to 192 nodes and 216 tasks).

We investigate two real-world applications. In both examples, we are given a data-driven task graph and a set of nodes. The data-driven task graph is then preprocessed by the *Srijan macroprogramming toolkit* [11] to provide a set of constraints for each node, where tasks can be run and what are the possible data flows. The goal is then to find a mapping that satisfies these constraints and minimises the energy consumption of the network.

The first application is a highway traffic management system where the aim is to reduce the congestion of vehicles on a highway by controlling speed limits and vehicle access to the highway. Figure 1 represents one sector of a highway. There are uniformly distributed forwarding nodes on the edge of the highway to

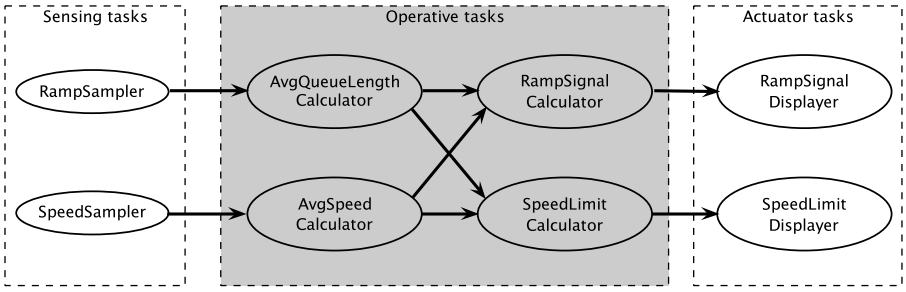

Figure 2. Specification of data-driven task graph for *one* sector of a highway in the traffic problem

let the sectors communicate. Speed sensors are randomly distributed on the four lanes so that each is in the range of at least another speed sensor or a forwarding node. The speed sensor on each lane senses the speed of passing vehicles, a presence sensor indicates the presence of a vehicle on the ramp, and a red/green signal on the ramp controls vehicle entry to the highway. Figure 2 depicts a specification of the data-driven task graph for one *sector* of a highway. Vertices in this graph represent tasks and arcs indicate flow of data between the tasks. The operative tasks are highlighted by grey colour and are free to map to any nodes. The tasks `SpeedSampler` and `RampSampler` use a sensor to capture the speed or presence of a vehicle, respectively. The sensing tasks send the sensed data to the operative tasks. There are four operative tasks: the `AvgQueueLengthCalculator` takes the presence data from `RampSampler` tasks at three consecutive sectors, and computes the average number of vehicles waiting in a queue per round. The `AvgSpeedCalculator` takes the speed of vehicles from `SpeedSampler` tasks at three consecutive sectors, and computes the average speed of vehicles on each lane. The `RampSignalCalculator` takes the average queue length and average speed of vehicles from three consecutive sectors and decides whether it is necessary to use the ramp signal or not. The `SpeedLimitCalculator` takes the average queue length and average speed of vehicles from three consecutive sectors and calculates the speed limit, which is sent to the `SpeedLimitDisplayer` to show on the display. The `RampSignalDisplayer` takes the data from the `RampSignalCalculator` and turns on the red or green signal accordingly. We experiment on realistic instances of this problem ranging from 7 nodes and 9 tasks up to 150 nodes and 216 tasks.

The second application is building environment management for the monitoring of heating, ventilation, and air conditioning (HVAC). The sensing tasks are humidity sampling and temperature sampling, which are fixed to some nodes, and the actuator tasks are fixed to selected nodes to control humidity and temperature. The operative tasks collect the sampled data from sensors, compute averages, and respond with proper actions for the actuator tasks. We experiment

on realistic instances of this problem ranging from 7 nodes and 6 tasks up to 192 nodes and 144 tasks.

The operative tasks defined for the two applications are free to be mapped to any nodes. We minimise the energy consumed by each node, based on the communication cost for gathering the collected data from all the other nodes, processing the operations, and sending the results to the actuators. Note that, once tasks are mapped to nodes, we can compute the energy spent by each node in one round.

3 Model

After presenting the mathematical model of [13] for the task mapping problem,[1] we compare the mixed integer programming (MIP) implementation in [13] of this model with our much smaller and more direct constraint programming (CP) implementation thereof. We achieve this by exploiting a standard distinguishing feature of CP, namely the ability of indexing a matrix by decision variables, thereby referring to an unknown element of the matrix, using the *element* constraint [15].

3.1 Mathematical Model

Throughout this paper, the decision variables have uppercase identifiers, the constants have lowercase identifiers, the indices i and j refer to tasks, while the indices p, q, and r refer to nodes:

- Let n be the number of nodes, the nodes being numbered $0, 1, \ldots, n-1$.
- Let t be the number of tasks, the tasks being numbered $0, 1, \ldots, t-1$.
- Let a be the set of arcs in the task graph, with arc (i, j) indicating that task i is sending data to task j.
- Let $f[i]$ be the firing rate of task i, that is the number of times it is fired at each round.
- Let $s[i, j]$ be the number of units of data sent from task i to task j.
- Let $e[q, r, p]$ be the energy consumed by node p while routing one unit of data that is sent from node q to node r.
- Let $e^0[p]$ be the initial energy of node p.
- Let $N[i]$ denote the node that task i is mapped to, with $0 \leq i < t$ and $0 \leq N[i] < n$.
- Let $E[p]$ denote the energy spent by node p according to the tasks mapped to it by the $N[i]$ decision variables, with $0 \leq p < n$ and $E[p] \in \mathbb{N}$.

We now introduce the constraints and the objective function.

[1] Actually, we present its simplification to the scenario used in its experiments, namely under the assumption that computation energy is negligible compared to communication energy.

As explained in Section 2, a sensing or actuator task may be restricted to a suitable subset of the nodes, and therefore it cannot be mapped to any other node. Such placement constraints are modelled as follows:

$$N[i] \neq p \qquad \text{for every task } i \text{ that cannot be mapped to node } p \tag{1}$$

The energy $E[p]$ spent by node p in one round is the sum of its energies consumed for routing data between every pair of nodes that are communicating via p:

$$E[p] = \sum_{(i,j)\in a} f[i] \cdot s[i,j] \cdot e[N[i], N[j], p] \qquad (0 \leq p < n) \tag{2}$$

Note that the indices $N[i]$ and $N[j]$ on the e matrix in (2) are decision variables.

The objective is to minimise the maximum energy spent by the nodes in one round. Indeed, in task mapping for WSNs, we aim at maximising the time-to-reconfiguration, which is the time when the energy level of some node goes below some fraction of its initial energy. Thus, the objective function to be minimised is the following:

$$\max_{0\leq p<n} \frac{1}{e^0[p]} \cdot E[p]$$

To perform this minimisation, we introduce an integer decision variable $C \in \mathbb{N}$ and constrain the fraction of initial energy spent in each round by node p to be at most C, subject to minimising C:

$$\frac{1}{e^0[p]} \cdot E[p] \leq C \qquad (0 \leq p < n) \tag{3}$$

Since all initial energies are the same in our datasets, we simplify this to:

$$E[p] \leq C \qquad (0 \leq p < n) \tag{4}$$

3.2 The MIP Implementation

Due to the $N[i]$ and $N[j]$ decision variables among the indices on the e matrix in the energy constraints (2), the latter have to be reformulated toward usage by a MIP solver. We first replace the array N of t decision variables of the mathematical model by a matrix X of $t \cdot n$ Boolean (0/1) decision variables, such that $X[i,p] = 1$ if and only if task i is mapped to node p (that is, if and only if $N[i] = p$), with $0 \leq i < t$ and $0 \leq p < n$, under the additional constraints that every task i is mapped to exactly one node p:

$$\sum_{p=0}^{n-1} X[i,p] = 1 \qquad (0 \leq i < t)$$

The placement constraints (1) then become as follows:

$$X[i,p] = 0 \qquad \text{for every task } i \text{ that cannot be mapped to node } p$$

The energy constraints (2) can then consider all possible nodes q and r and multiply $e[q,r,p]$ by $X[i,q] \cdot X[j,r]$ to indicate whether there is a message routed from q to r via node p:

$$E[p] = \sum_{(i,j) \in a} \sum_{q=0}^{n-1} \sum_{r=0}^{n-1} f[i] \cdot s[i,j] \cdot e[q,r,p] \cdot X[i,q] \cdot X[j,r] \qquad (0 \leq p < n)$$

However, the factor $X[i,q] \cdot X[j,r]$ makes this a non-linear constraint. To linearise it, following [12] as in [13], we add an array Y of $t^2 \cdot n^2$ redundant Boolean (0/1) decision variables, such that the following channelling constraints enforce that $Y[i,q,j,r] = X[i,q] \cdot X[j,r]$:

$$\begin{array}{r} Y[i,q,j,r] \leq X[i,q] \\ Y[i,q,j,r] \leq X[j,r] \\ Y[i,q,j,r] \geq X[i,q] + X[j,r] - 1 \end{array} \qquad (0 \leq i,j < t,\ 0 \leq q,r < n)$$

The energy constraints (2) now become as follows:

$$E[p] = \sum_{(i,j) \in a} \sum_{q=0}^{n-1} \sum_{r=0}^{n-1} f[i] \cdot s[i,j] \cdot e[q,r,p] \cdot Y[i,q,j,r] \qquad (0 \leq p < n)$$

The objective remains the minimisation of C, subject to (4).

In summary, we have replaced t integer decision variables by $t^2 \cdot n^2 + t \cdot n$ Boolean decision variables and added $3 \cdot t^2 \cdot n^2 + t$ constraints to the mathematical model. This is not negligible, as WSNs may have hundreds of nodes and hundreds of tasks. Further, the single summation (2) has become a triply nested summation.

3.3 The CP Implementation

A CP model can much more directly implement the mathematical model, due to its unique ability of indexing arrays and matrices by decision variables. The $element(A, I, V)$ constraint [15] holds if and only if $A[I] = V$, where A is an array of constants or decision variables, I is a linear expression on integer decision variables, and V is a constant or decision variable.

Let e^p designate the array obtained by flattening row-wise the two-dimensional matrix slice $e[*,*,p]$ of the three-dimensional matrix e, that is the following relationship (not constraint) holds:

$$e^p[q + n \cdot r] = e[q,r,p] \qquad (0 \leq p,q,r < n) \tag{5}$$

Let function f map the edges of the task graph a to the integers $0, 1, \ldots, |a| - 1$. We first add an array Z of $|a| \cdot n$ redundant integer decision variables to the mathematical model, such that $Z[f(i,j),p]$ denotes $e[N[i],N[j],p]$. Due to (5), this relationship can be enforced by the following channelling constraints:

$$element(e^p, N[i] + n \cdot N[j], Z[f(i,j),p]) \qquad (0 \leq p < n,\ (i,j) \in a)$$

on which domain consistency (all values not participating in a solution are pruned) can be achieved efficiently. The energy constraints (2) now become as follows:

$$E[p] = \sum_{(i,j)\in a} f[i] \cdot s[i,j] \cdot Z[f(i,j),p] \qquad (0 \le p < n)$$

The objective remains the minimisation of C, subject to (4).

In summary, we have added $|a| \cdot n$ integer decision variables and $|a| \cdot n$ constraints to the mathematical model. Since $|a| = O(t^2)$, we have added $O(t^2 \cdot n)$ integer decision variables and $O(t^2 \cdot n)$ constraints, hence at least n times fewer additional decision variables and constraints than in the MIP model. Further, note that the energy constraints of the CP model have a *single* summation, like (2) in the mathematical model, and unlike the *triply* nested summation in the MIP model.

In the *Comet* [16] CP modelling language, one can literally write the energy constraints as in (2), though another domain-consistent implementation of such a multi-dimensional *element* constraint is made.

Another distinguishing feature of CP is the ability to define a custom search procedure for the model at hand, rather than having to rely on a fixed search procedure like MIP solvers. In CP, complete tree search is interleaved with *propagation* (the usually polynomial-time elimination of provably impossible values from the domains of the decision variables) at every node. Search is parameterised by heuristics that specify what decision variable, value, and comparison operator is selected for branching decisions. Search usually does *not* branch over all decision variables, as some decision variables will eventually take their values just through propagation of the channelling constraints when branching on decisions involving the other decision variables.

Our search procedure branches on the $N[i]$ decision variables *only*. It selects a most constrained decision variable $N[i]$ (that is variable ordering `INT_VAR_DEGREE_MAX` in *Gecode* [5]), and for tie-breaking, it selects the left-most decision variable with the currently smallest domain (`INT_VAR_SIZE_MIN` in *Gecode*). For branching on the selected decision variable $N[i]$, we consider the following situation (known as `INT_VAL_RANGE_MAX` in *Gecode*):

1. If the domain of the selected decision variable has a single range $\ell \ldots u$, then we try the decision $N[i] > \lfloor \frac{\ell+u}{2} \rfloor$ on the left branch and the decision $N[i] \le \lfloor \frac{\ell+u}{2} \rfloor$ on the right branch.
2. Otherwise, we try the left-most largest range on the left branch and the rest of the domain on the right branch.

4 Experiments

We experimented with various realistic instances (chosen from [13], plus larger ones produced by the instance generator used in [13]) for the two applications mentioned in Section 2, namely the highway traffic management and HVAC

problems. For each of the two applications, we start from the smallest size (highway traffic $\langle n, t\rangle = \langle 7, 9\rangle$ and HVAC $\langle 7, 6\rangle$) and increase the instance size by at least 10 nodes at each step up to the largest size (highway traffic $\langle 150, 216\rangle$ and HVAC $\langle 192, 144\rangle$).

Our CP model is implemented in *Gecode* [5] (revision 3.4.0) and runs under Mac OS X 10.6.4 64 bit on an Intel Core 2 Duo 2.53 GHz with 3MB L2 cache and 4GB RAM. We set a timeout of 600 second for each instance, recording the time at two points: the time to optimally solve the instance and the time at which the minimum cost has been reached.

We also solved our instances using the MIP solvers *Gurobi* (revision 3.0.1),[2] *SCIP* (revision 1.2.0),[3] and *lp_solve* (revision 5.5),[4], under their default parameters, the same system configuration, and the same experimental set-up. We chose *lp_solve* only since it is the solver used in [13]. We chose to experiment also with *Gurobi* and *SCIP*, as these two solvers are among the fastest commercial and non-commercial MIP solvers, respectively (according to the *SCIP* home page).

The instance data are stored both in an internal format used by our CP model and in the MPS file format [11]. MPS is a standard column-oriented format for storing linear programming models for MIP solvers.

In Tables 1 and 2, we present the results of solving highway traffic management and HVAC instances using our CP model compared to the model solved by the three MIP solvers. For highway traffic management, *Gecode* always proves the optimum solution faster than *Gurobi*, and specifically, for larger instances, *Gecode* proves the optimum solution at least an order of magnitude faster than *Gurobi*. For HVAC, in most of the instances *Gecode* is competitive with *Gurobi*, which is remarkable as *Gecode* is a non-commercial solver.

These results show that constraint programming can efficiently solve a combinatorial problem introduced in task mapping for WSNs. It is also worth mentioning that MIP solvers use a pre-solve phase to eliminate as many variables and constraints as possible before solving the actual problem. For example, pre-solve in one of the hardest instances (highway traffic $\langle 124, 60\rangle$) using *Gurobi* is taking 22 seconds, which are included in the runtime of 165 seconds in Table 1, and this is already longer than the time spent by *Gecode* to find the same solution.

5 Conclusion

Macroprogramming for wireless sensor networks (WSNs) is an evolving area, where efficiency of a WSN can benefit considerably by task mapping in a configuration phase, as well as in reconfiguration when the energy level of a node drops below some amount.

[2] available from http://gurobi.com
[3] available from http://scip.zib.de
[4] available from http://lpsolve.sourceforge.net

Highway	*Gecode*			*Gurobi*			*SCIP*			*lp_solve*		
$\langle n, t \rangle$	time	time_{opt}	cost	time	time_{opt}	cost	time	time_{opt}	cost	time	time_{opt}	cost
$\langle 7, 9 \rangle$	0.001	**0.010**	20	< 1	0.03	20	7.56	7.56	20	< 1	0.24	20
$\langle 13, 18 \rangle$	0.009	**0.024**	60	< 1	0.42	60	70.50	70.83	60	1	27.14	60
$\langle 19, 27 \rangle$	0.022	**0.034**	100	< 1	8.12	100	58.70	> 600.00	100	3	> 600.00	100
$\langle 25, 36 \rangle$	0.049	**0.060**	100	< 1	10.81	100	168.00	> 600.00	100	10	> 600.00	100
$\langle 32, 45 \rangle$	0.091	**0.109**	100	< 1	7.48	100	146.00	> 600.00	100	240	> 600.00	100
$\langle 38, 54 \rangle$	0.166	**0.222**	100	< 1	11.07	100	106.00	> 600.00	172	13	> 600.00	150
$\langle 44, 63 \rangle$	0.264	**0.300**	100	< 1	45.50	100	167.00	> 600.00	148	21	> 600.00	180
$\langle 63, 90 \rangle$	0.985	**1.048**	100	98	153.97	100	450	> 600.00	120	68	> 600.00	230
$\langle 74, 36 \rangle$	0.549	> 600.000	300	38	> 600.00	300	165.00	> 600.00	480	> 600	> 600.00	–
$\langle 75, 108 \rangle$	1.888	**2.007**	100	142	428.90	100	459	> 600.00	179	270	> 600.00	210
$\langle 88, 126 \rangle$	3.350	**3.499**	100	1	117.80	100	536	> 600.00	177	> 600	> 600.00	–
$\langle 100, 144 \rangle$	5.359	**5.693**	100	2	119.09	100	335	> 600.00	189	> 600	> 600.00	–
$\langle 113, 162 \rangle$	8.427	**8.756**	100	2	96.73	100	392	> 600.00	207	> 600	> 600.00	–
$\langle 124, 60 \rangle$	3.155	**286.338**	300	165	> 600.00	300	> 600.00	> 600.00	–	> 600	> 600.00	–
$\langle 125, 180 \rangle$	12.545	**12.956**	100	3	329.46	100	301	> 600.00	209	> 600	> 600.00	–
$\langle 138, 198 \rangle$	17.598	**18.282**	100	421	546.33	100	31.5	> 600.00	240	> 600	> 600.00	–
$\langle 150, 216 \rangle$	24.205	**25.033**	100	3	> 600.00	100	> 600.00	> 600.00	–	> 600	> 600.00	–

Table 1. Results of different solvers for realistic instances of the highway traffic management task mapping problem. The time unit is seconds, in the precision reported by the respective solver. A 600-second-timeout was used and time is recorded at two points: the time to optimally solve the instance, denoted by time_{opt}, and the time to reach the minimum cost, denoted by 'time'. The runtime was rounded by the solver to 0 for some instances: we report it to be '< 1'. In some instances, the solver failed to reach any cost before the timeout, which is shown by '–' in the table. The boldface values indicate best times to prove optimality for the corresponding instance.

5.1 Summary

We presented a constraint programming (CP) model for the task mapping problem in a multi-hop heterogeneous WSN. Our model involves placement constraints and the cost of routing. The optimisation is done by minimising the maximum energy spent by each node in each round. We have shown that our CP model is at least competitive with a published mixed integer programming (MIP) model, if not outperforming it by at least one order of magnitude.

5.2 Related Work

The main related work to ours is [13], where the problem specification and formulation are taken from, and where the remaining related work is discussed;

HVAC	*Gecode*			*Gurobi*			*SCIP*			*lp_solve*		
$\langle n, t \rangle$	time	time_{opt}	cost	time	time_{opt}	cost	time	time_{opt}	cost	time	time_{opt}	cost
$\langle 7, 6 \rangle$	0.110	0.110	10	< 1	$<$ **0.01**	10	0.10	0.13	10	< 1	< 0.01	10
$\langle 11, 9 \rangle$	0.001	0.012	60	< 1	$<$ **0.01**	60	0.10	0.15	60	< 1	0.02	60
$\langle 23, 8 \rangle$	0.003	0.013	480	< 1	**0.01**	480	0.28	0.28	480	< 1	0.34	480
$\langle 34, 26 \rangle$	0.011	0.032	1080	< 1	**0.02**	1080	0.50	0.50	1080	2	2.43	1080
$\langle 43, 33 \rangle$	0.030	0.060	1980	< 1	**0.03**	1980	0.50	0.55	1980	3	8.56	1980
$\langle 54, 41 \rangle$	0.065	0.106	3080	< 1	**0.06**	3080	0.80	0.80	3080	9	20.61	3080
$\langle 66, 50 \rangle$	0.136	0.231	4760	< 1	**0.11**	4760	1.19	1.19	4760	14	41.01	4760
$\langle 74, 56 \rangle$	0.208	0.361	6080	< 1	**0.18**	6080	1.50	1.50	6080	37	86.22	6080
$\langle 86, 65 \rangle$	0.392	0.645	8360	< 1	**0.37**	8360	2.10	2.16	8360	55	137.03	8360
$\langle 97, 73 \rangle$	0.573	1.027	10500	< 1	**0.59**	10500	2.79	2.79	10500	106	309.52	10500
$\langle 106, 80 \rangle$	0.821	1.490	12960	< 1	**0.81**	12960	3.31	3.31	12960	247	> 600.00	12960
$\langle 192, 144 \rangle$	8.267	13.555	43120	3	**3.90**	43120	15.7	15.76	43120	> 600.00	> 600.00	–

Table 2. Results of different solvers for realistic instances of the HVAC task mapping problem. The time unit is seconds, in the precision reported by the respective solver. A 600-second-timeout was used and time is recorded at two points: the time to optimally solve the instance, denoted by time_{opt}, and the time to reach the minimum cost, denoted by 'time'. The runtime was rounded by the solver to 0 for some instances: we report it to be '< 1'. In some instances, the solver failed to reach any cost before the timeout, which is shown by '–' in the table. The boldface values indicate best times to prove optimality for the corresponding instance.

we took the instances from that work to compare with MIP solvers. However, we use CP to capture directly the mathematical model.

In [14], the authors only investigate *single*-hop *homogeneous* WSNs. We consider *multi*-hop *heterogeneous* networks including routing costs. That work, unlike ours, also considers the minimum schedule length subject to energy consumption constraints.

To the best of our knowledge, there has been no work addressing a CP approach for a general case of task mapping in a multi-hop WSN achieving energy optimisation under routing costs.

The closest CP approach to our work is [10], where the problem is the optimal deployment of eventually serialisable data services (ESDS) in distributed systems. The ESDS problem includes separation constraints (every pair in a subset of modules must not be mapped to the same server) and co-location constraints (every pair in a subset of modules must be mapped to the same server), unlike our WSN task mapping problem. The ESDS communication cost (the number of hops) is paid only by the end-points, whereas our work considers the communication cost that any particular node pays for routing a message between any two other nodes. Whereas in ESDS one minimises the *sum* of all hops required to send messages between modules, we minimise the *maximum* energy spent by

any node. Even if we also minimised the total energy spent by all nodes, it would still be necessary to have the initial-energy constraints (3), which do not occur in ESDS. They also report impressive speed-ups over a MIP model of their CP model, which also uses a multi-dimensional *element* constraint, and the gap to the MIP model is probably made bigger by the *allDifferent* constraints capturing the extra separation constraints. They also show that dynamic symmetry breaking can further improve the CP model.

5.3 Future Work

We expect that a more advanced branching that takes the task graph and the target network features into account may result in a faster solution or a better cost under the 600-second-timeout.

In [13], two objectives are used: minimising the maximum energy spent by all nodes, namely *energy balance*, and minimising the sum of the energy spent by all nodes. Energy balance is a better metric, since it maximises the time to reconfiguration. Minimising the total energy spent is the classical metric. We can also consider minimising the total energy as our cost metric and investigate the effect of this classical metric.

The work in [13] also addresses a scenario where *multiple paths* are possible between nodes, which further increases the complexity of the problem, and is considered as our future work. The computation cost of tasks was ignored both in our CP model and in the experiments with the MIP model of [13], since the computation cost for invoking a task by a node, in our data set, is far smaller than the communication cost of routing the message. However, it might be of interest to investigate the impact of computational cost on the model. It is also interesting to compare our CP model to the greedy heuristic introduced in [13].

An alternate approach in solving a combinatorial problem such as task mapping is local search. It may be more efficient to find a reasonably good solution according to the objective function using local search, however it may not be possible to solve optimally the problem.

Acknowledgements. This research is sponsored by the Swedish Foundation for Strategic Research (SSF) under research grant RIT08-0065 for the project *ProFuN: A Programming Platform for Future Wireless Sensor Networks*. We thank Animesh Pathak of INRIA Paris-Rocquencourt (France) for useful discussions and the datasets.

References

1. I. F. Akyıldız, W. Su, Y. Sankarasubramaniam, and E. Çayırcı. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, 2002.
2. A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner. The abstract task graph: A methodology for architecture-independent programming of networked sensor systems. In *EESR'05: Proceedings of the 2005 workshop on End-to-End, Sense-and-Respond systems, applications and services*, pages 19–24. USENIX Association, 2005.

3. M. Demirbaş. Wireless sensor networks for monitoring of large public buildings. *Computer Networks*, 46:605–634, 2005.
4. S. C. Ergen and P. Varaiya. Energy efficient routing with delay guarantee for sensor networks. *Wireless Networks*, 13(5):679–690, 2007.
5. Gecode Team. Gecode: Generic constraint development environment, 2006. Available from http://www.gecode.org/.
6. R. Gummadi, O. Gnawali, and R. Govindan. Macro-programming wireless sensor networks using KAIROS. In V. K. Prasanna, S. S. Iyengar, P. G. Spirakis, and M. Welsh, editors, *Distributed Computing in Sensor Systems (DCOSS), First IEEE International Conference*, volume 3560 of *Lecture Notes in Computer Science*, pages 126–140. Springer Verlag, 2005.
7. T. T. Hsieh. Using sensor networks for highway and traffic applications. *IEEE Potentials*, 23(2):13–16, April–May 2004.
8. D. Kumar, T. C. Aseri, and R. Patel. Energy efficient heterogeneous clustered scheme for wireless sensor networks. *Computer Communications*, 32(4):662–667, 2009.
9. A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA'02: Proceedings of the 1st ACM international workshop on Wireless Sensor Networks and Applications*, pages 88–97. ACM Press, 2002.
10. L. Michel, A. A. Shvartsman, E. L. Sonderegger, and P. Van Hentenryck. Optimal deployment of eventually-serializable data services. In L. Perron and M. A. Trick, editors, *Proceedings of CP-AI-OR'08*, volume 5015 of *LNCS*, pages 188–202. Springer-Verlag, 2008.
11. J. L. Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, 1987.
12. G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd. *Optimization*. North-Holland, Elsevier, 1989.
13. A. Pathak and V. K. Prasanna. Energy-efficient task mapping for data-driven sensor network macroprogramming. *IEEE Transactions on Computers*, 59(7):955–968, July 2010.
14. Y. Tian, E. Ekici, and F. Özgüner. Energy-constrained task mapping and scheduling in wireless sensor networks. In *IEEE International Conference on Mobile Ad hoc and Sensor Systems*, pages 8–218. IEEE Computer Society Press, 2005.
15. P. Van Hentenryck and J.-P. Carillon. Generality versus specificity: An experience with AI and OR techniques. In T. Mitchell and R. Smith, editors, *Proceedings of AAAI'88*, pages 660–664. AAAI Press, 1988.
16. P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*. The MIT Press, 2005.