Be strong and courageous. Do not be afraid, and do not be dismayed.

— יְהוָה אֱלֹהָי 1250 B.C.

University of Alberta

Time and Throughput Efficient Scheduling for Data Gathering in Wireless Sensor Networks

by

Evandro de Souza

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2013
Edmonton, Alberta

To the *Only One* worthy
to receive glory and honor and power

Abstract

Wireless sensor networks have become a very important tool for monitoring physical and environmental conditions over a wide area. These networks are distributed collections of small sensor nodes. Typically, sensor nodes collect data that must converge to a single sink location, possibly across multiple hops. The data on the way to sink location can be aggregated inside the network. This aggregation convergecast process requires significant coordination between the sensor nodes in the selection of routes, and in the scheduling of the times for data transmission. Both impose considerable restrictions on the communication protocols.

In this thesis, we study problems of routing and scheduling in wireless sensor networks when *precedence* and *resource constraints* requirements are present. Instead of looking for a schedule and logical topology that have been designed for a generic network or traffic demand, we study collision-free scheduling and logical topology solutions for applications restricted by both constraints.

We propose a model for the aggregation convergecast problem using constraint satisfaction to extract characteristics of optimal solutions and to expose the limitations of current solutions in the literature. Using the characteristics observed, we propose the construction of a logical topology that balances the effects of both constraints on the data collection schedule length. A typical solution for the problem encompasses two phases, a initial aggregation tree selection, followed by the node transmission schedule. We show that the scheduling part can be modeled as a Mixed Graph Coloring, and we propose a scheduling solution.

Departing from the emphasis on schedule length (delay) minimization, we study the problem of throughput-oriented solutions, where the data collection rate is of higher importance, instead of delay. We relax the restriction that all precedence between nodes must be satisfied within a single collection period, and use pipelining to increase the data collection throughput.

Acknowledgements

The successful conclusion of this PhD is not fruit of the isolated effort of a single person, but it is the combined investment of several people, even before it started.

Foremost, I acknowledge the effort and dedication of my family: my parents, my wife and son. My father (Delci Barbosa de Souza) and my mother (Eda Maiolino de Souza) never faulted on providing me with the stimulus and resources to continue my studies. Their dedication is still a solid example to me. My wife Nilcéia is the best person I ever met. Her unconditional love and support made it possible for me to achieve today's success. She shared my achievements and failures. Her unconditional support strengthened me through the ups and downs of this journey. My son Gabriel is my reminder that life is not only study and work.

I would like to express my deepest gratitude to my advisors Dr. Ioanis Nikolaidis and Dr. Pawel Gburzynski for their excellent supervision throughout the years. The countless hours of discussion and paper revision, as well as out-of-the-box thinking, bore several fruits, and finally, the completion of my Thesis. I extend my gratitude to the supervisory and examining committee members.

I am also thankful to several fellow PhD students that directly or indirectly helped me during my PhD program: Amin Jorati, Baljeet Malhorta, Benyamin Shimony, Feng Chen, Filipe Mesquita, Ken Bauer, Levi Santana, Nicholas Boers, Sajib Barua, and Saman Vaisipour.

Finally, I would like to thank the University of Alberta who provided me with financial support and opportunity as Teaching Assistant. The contact with undergraduate students and experienced professors enriched my professional experience.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ACS	Aggregation Convergecast Scheduling
ACSPIPE	Aggregation Convergecast Scheduling Pipeline
BDMRST	Bounded-Degree Minimum Radius Spanning Tree
BFS	Breadth-First search
BSPT	Balanced Shortest Path Tree
CBS	Constraint-Based Scheduling
CDS	Connected Dominating Set
COP	Constraint Optimization Problem
CP	Constraint Programming
CPU	Central Processing Unit
CSP	Constraint Satisfaction Problem
DAG	Directed Acyclic Graph
DAS	Distributed Aggregation Scheduling
DFS	Depth-First Search
GECODE	Generic Constraint Development Environment
ILP	Integer Linear Problem
LAST	Light Approximation Shortest Path Tree
LDS	Limited Discrepancy Search
LSP	Longest Shortest Path
MAC	Media Access Control
MDB	Minimum Delay Broadcast
MCA	Minimum Cost Arborescence
MGC	Mixed Graph Coloring
MIMO	Multiple Input Multiple Output
MINT	Minimum Interference Network Topology
MIT	Minimum Interference Tree
MTBE	Methyl Tert-Butyl Ether
MTS	Multiprocessor Task Scheduling
NP	Nondeterministic Polynomial Time
PAS	Postorder Aggregation Scheduling

SDA	Shortest Data Aggregation
SISO	Single Input Single Output
SLT	Shallow Light Tree
SPT	Shortest Path Tree
TDMA	Time-Division Multiple Access
TS	Tree Size
UDG	Unit Disk Graph
X-MAC	Low Power MAC
WIRES	Weighted Incremental Ranking for convergEcast with aggregation Scheduling
WSN	Wireless Sensor Networks

Chapter 1

Introduction

1.1 Wireless Sensor Network Applications

Wireless sensor networks (WSN) are distributed collections of small devices. Each device is equipped with a processing capability to execute small applications, a wireless radio transceiver to exchange information with other devices, and sensors to collect information about the environment. Sensor network applications include military sensing, physical security, environment monitoring, building and structures monitoring, among others [29].

By their nature, wireless sensor networks have resource and performance constraints. The nodes are usually battery powered, which imposes a severe energy restriction to the applications. The processor usually also has a limited processing capability. Wireless communication has its own limitations. Multi-hop communication may be required to compensate for a node's short transmission range, which increases the possibility of congesting the wireless media.

These limitations demand an integrated design approach, a combination of algorithms and protocols such that the use of scarce resources can be optimized toward a specific goal. Therefore, customized solutions are desirable, instead of using solutions optimized for an unspecified communication pattern [40, 77, 84] or a generic data demand [21].

Let us consider the case of monitoring a region subject to fast changing environmental conditions, like when background radiation level increases, indicating the presence of radiation sources [80, 90] or when a contaminant spill alert is triggered and its underground transport has to be monitored [11, 42] (exemplified on Figures 1.1 and 1.2). Under these circumstances, a monitoring network has to be designed with specific requirements in mind. A

minimum delay data collection may be necessary for analysis and rapid response; wireless may be the only possible way to interconnect monitoring sensors to the collecting point; very little or no mobility is required from the sensors, which are placed in predetermined points on the monitored area or surrounding a possible toxic contaminant area; a continuous data flow may be necessary to be collected for analysis; in-network aggregation of sensed data (combination of flows of information in intermediate nodes before the collection point) may be required to reduce the amount of traffic and reduce the energy used in the transmissions; different queries may be issued to evaluate the extent of affected areas and any specifics of the hazard condition (ex. pH, redox condition in groundwater, etc); and may not be necessary to know the exact reading of a specific sensor, but an average or the maximum value over a region.

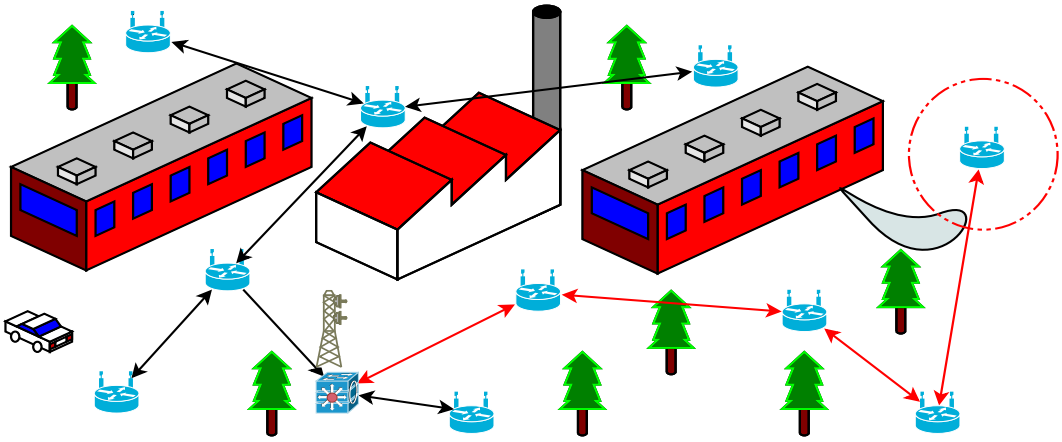


Figure 1.1: Factory Sensor Net

The same scenario may require an application with a different set of requirements and goals. Instead of collecting data in the minimum amount of time, an application may be interested in taking periodic samples of the monitored area, transmit them to a central controller, where the variation of the observed condition are reconstructed and analyzed. The quality of the reconstruction does not rest more on the collection delay, but on the throughput of field measurement snapshots arriving at the central controller. Data aggregation can still be used, however the system optimization criteria and requirements are different.

Using the previous scenario and applications as example, one could design an algorithmic solution for data collection, using the same design for applications with different goals, without understanding which constraints are appropriate for each application, and how the outcome would be affected. This understanding is especially relevant for problems whose solution encompasses more than one aspect, as is the case with network application, where

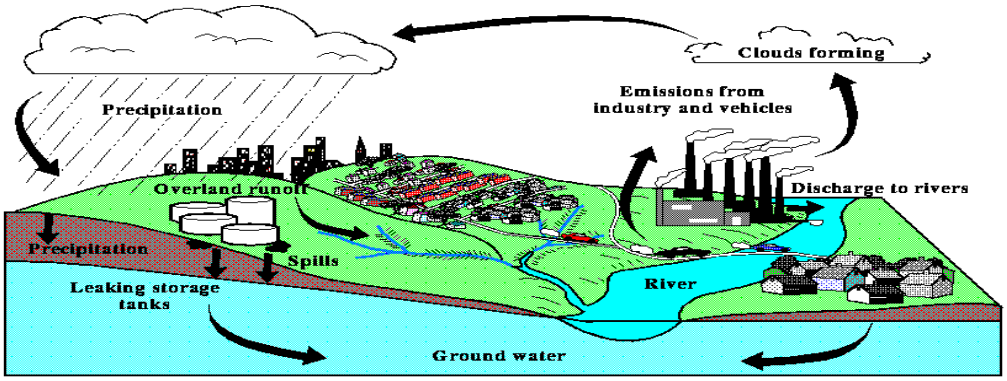


Figure 1.2: The Movement of MTBE in the Environment
Taken from [1]

a joint routing and scheduling solution is required.

Similarly, one could also design a solution addressing only specific restrictions, without efficiently combining another relevant constraint. This biased design leads to inefficiencies, as one constraint has not been taken into account. This incomplete approach precludes the overall solution from achieving its best performance. For example, a selected logical topology may not work as well as it could, because it was selected to minimize the effect of only one constraint.

The existence of suitable models, with comprehensive and clear constraints, is a necessary tool to help design applications and evaluate their performance. Some research in wireless sensor networks is restricted to formally defining the problem, in some mathematical form (like Integer Linear Problem (ILP)), without comparing the model output with the output produced by their proposed algorithm [26, 76].

The environment monitoring example presented before calls for distinct requirements: (1) the use of some form of in-network data aggregation to decrease the amount of data transmitted; (2) keeping the schedule length short, for the sake of quick application response, or for the sake of collection throughput; (3) the need of a balanced routing and scheduling solution, because of their interdependency. The importance of the *all-to-one* communication pattern in wireless network applications, the incomplete understanding of the constraints restricting data collection performance, and the challenge of balancing routing and scheduling in a single solution raised our interest in aggregation convergecast. It clearly combines the selection of a suitable logical topology (routing), with the selection of a feasible node trans-

mission schedule (scheduling), while having to satisfy a set of different, and sometimes conflicting, constraints.

1.2 Previous State-of-Art

We found some limitations on the literature throughout the course of our research about aggregation convergecast applications. The main limitations are described next.

- **Absence of suitable models**

  Even though some effort has been done in the literature to provide a mathematical model [48, 76] for aggregation convergecast, studies fall short to effectively use the proposed models. A thorough investigation of the problem using such proper and complete models could show hidden characteristics, and expose how the outcome of the aggregation convergecast would be affected in face of changes in the design objectives.

- **Incomplete understanding of constraints**

  A direct consequence of the absence of suitable models is an incomplete understanding of how and, in which circumstances, each constraint influences the solution. Hence, a solution that addresses only one constraint may lead to results far from the best possible solution that a problem can offer; or, in circumstances where its influence is smaller, another influential constraint is overlooked. One example of this limited approach is the use of the precedence constraint (when a node only transmits after has received from all its children) at expense of the resource constraint (interference for wireless networks).

- **Restricted framework to design solutions**

  The incomplete understanding of the constraints induces a restricted framework to design solutions. If the importance of a type of constraint is lessened, only frameworks that accentuate the other constraint are used. In our problem, this had happened with precedence and resource constraints. Likewise, if one component of a joint routing and scheduling solution is given preference over another part, the design is restricted only to this part of the solution. This bias can be seen in the solutions for aggregation convergecast in the literature. Virtually all studies propose solutions based on two phases approach, each phase favoring one of the components of the solution (aggregation tree and transmission scheduling). The majority of the studies address

the routing part first. No attempt was done to see the outcome of the problem if the scheduling part is addressed first, and its impact on the results. Besides, as the problem is partitioned in two phases, there is a possibility that each phase be modeled separately, and studied as an independent problem, even though they are not independent.

- **No Use of Parallelism**

  The solutions found in the literature exploit any available opportunity to use *spatial parallelism* of the transmissions as a form of increasing the throughput and reducing delay. However, the use of *temporal parallelism*, such as **pipelining**, is rarely mentioned. Pipelining can be understood as a form of temporal concurrency, where different nodes in a path toward the sink can transmit at the same time data from separate collection *rounds*. This form of parallelism has the potential of increasing even more the application performance under certain circumstances.

Our research is a step forward in addressing these limitations. First, it proposes a pervasive model using Constraint Programming [33], to express separately each constraint. Such model allows us to explore unclear aspects of the aggregation convergecast problem, and highlights some logical topology shortcomings. Our study also contributes by extending the classic conflict graph model [21] to attend the specific characteristics of the aggregation convergecast restrictions. Second, it addresses the effects of precedence and resource constraints when non-conservative flows (when packets are aggregated or multiplied during data collection) are present. We tackle both restrictions by balancing their influence on the creation of a logical topology (aggregation tree). Third, our research expands the solution frameworks by studying the effect of each restriction individually and their combination on the logical topology and on the scheduling. Another approach followed to broaden the solution options is by creating algorithms that address first the scheduling and subsequently the routing. Finally, our research proposes the use of pipelining in aggregation convergecast, eliminating the requirement of having a single snapshot being completely collected in a single schedule period.

1.3 Thesis Organization

We introduce in Chapter 2 notation, models, concepts and terminology used over the remaining of the thesis. We describe the concepts of in-network data processing, scheduling,

logical topology and wireless interference. We reveal the restrictions involved in developing a solution for aggregation convergecast, and the two elements by which a solution is composed. Criteria for evaluating results are presented in the end.

In Chapter 3, we model the routing and transmission scheduling problem for aggregation convergecast in wireless sensor networks as a Constraint Satisfaction Problem with the objective of obtaining the shortest possible schedule for a TDMA frame that allows the complete collection of data to the sink. All transmissions required for a completed data collection to the sink node are to be scheduled in a single TDMA frame. Constraints related to the topology, to the interference, and to the application logic, as well as their relationships are modeled. We compare the solutions found running our model in a constraint solver and the solutions found by existing algorithms in the literature.

Chapter 4 is dedicated to the creation of a solution that combines precedence constraints with the constraints caused by interference in shared wireless medium, expressed as resource constraints. In the first part, we propose an aggregation tree construction that is the synthesis of a tree tailored to precedence constraints and another tree tailored to resource constraints. In the second part, we show that the scheduling component of the aggregation convergecast can be modeled as a mixed graph coloring problem. In the mixed graph, arcs represent the precedence constraints and edges represent the resource constraints. The mixed graph chromatic number corresponds to the optimal schedule length. Bounds for the mixed graph coloring are provided and a branch-and-bound strategy is subsequently developed. We derive numerical results using the branch-and-bound strategy. The results allow a comparison against the current state-of-the-art heuristics for the problem.

In Chapter 5, we study the aggregation convergecast problem from a different perspective. In contrast to studies that attempt to minimize the data collection delay, we aim to increase the frequency (higher throughput) of snapshots received at the sink. To achieve higher throughput, we use a form of concurrent collection of multiple *snapshots* through the network, also known as pipeline. The use of pipeline forces us to abandon the usual restriction of requiring that all precedence constraint be satisfied in a single schedule period. This restriction is employed throughout the aggregation convergecast literature. In our solution, we opt for a scheduling formulation based on sink reachability constraints. Our solution reverses the usual order of steps found in existing algorithms, and produces a schedule before the construction of the logical tree topology. We compare our results against an algorithm that uses precedence constraints, and against another algorithm that

uses a variation of pipelining.

We conclude our thesis in Chapter 6. We present our contributions for knowledge advancement in the aggregation convergecast problem. Among the contributions are some insights that may require further investigation. The insights may open some exploration avenues about new ways to construct an algorithmic solution. One of these avenues was explored by Jakob's work [60], with our participation. He addressed the problem of finding a time efficient schedule and topology for aggregation convergecast in wireless sensor networks by reversing the bottom-up to a top-down approach. After explaining the results of Jakob's work, we present future directions that our research may take.

Appendix A groups the notation, definitions and algorithms of Graph Theory used in the thesis, serving us as guide reference and refresher to the reader.

Chapter 2

Background and Assumptions

2.1 Requirements and Constraints

Sensor network limitations require that applications be efficient according to some design objectives. The most critical objective is energy conservation. Other areas can be important as well, like collection latency, throughput, sensing coverage and path redundancy. In the case of the application presented in Chapter 1, there are two objectives that we want to address closely: first, the collection of information must be executed as quickly as possible (latency), and second, the collection must be performed as frequently as possible (throughput). The priority of each one is dependent of the application needs. One common application characteristic is the need for a continuous flow of information, instead of a isolated and single collection. The need for a continuous flow of information demands that the data be periodically sensed and collected.

Let us consider the specific scenario described in Chapter 1. In that scenario an application issues different requests and receives the corresponding responses. The application requires a combined design of logical topology and node transmission schedule.

It would be inefficient for the nodes to contact the collecting station directly for the response, yet it is important that information from each node is collected. Therefore, a node will need to transmit to one of its neighbors for its sensed information to reach the collecting node (see at Figure 2.1). This many-to-one communication paradigm, where data flows from many nodes towards a single point, is known as *convergecast* [35].

When the number of nodes is large, the amount of data generated might be voluminous, demanding significant communication and processing capability. Consequently, methods

for combining data at intermediate nodes are necessary for the reduction of the amount of data that needs to be forwarded to the sink.

This scheme where data is combined at intermediate nodes is called *in-network data processing*. This process allows intermediate nodes to execute (partially) this aggregation activity. If the information flows are merged or aggregated, they will not retain their original size. When information flows do not retain their original data volume we call then ***non-conservative flows***. This whole process of converging data to a single point and distilling data on the way is called *aggregation convergecast*.

The execution of data aggregation, closer to the data sources, is a interesting way to reduce data collection volume, especially when there are strong temporal and spatial correlation on the sensed data. The use of aggregation convergecast may have a significant impact on energy consumption and network efficiency, because it reduces the number of transmissions, increasing the expected network lifetime [43]. The reduction on the number of transmissions also results in shorter schedules, because there are fewer packet transmissions to be scheduled.

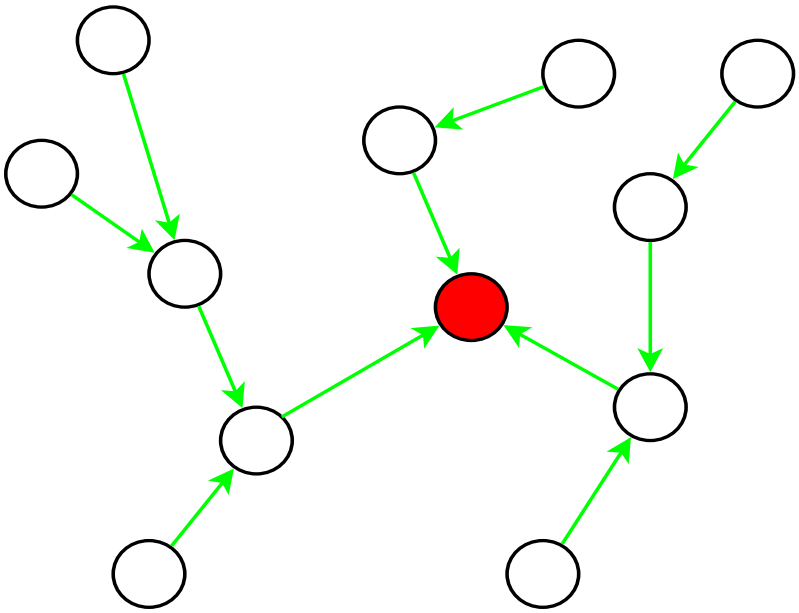


Figure 2.1: Convergecast Framework

In wireless sensor networks, the communication between two nodes is performed through wireless media. Technological restrictions may not allow that two nodes access the wireless media at the same time. This limitation is due to the interference caused by a node

transmission to other nodes not intended to be receiving this transmission. Interference is a form of ***resource constraint*** that limits the solution of aggregation convergecast.

An application may also have to ensure that all nodes in the network have opportunity to contribute with their sensed information without incurring into additional transmissions. This requirement can be fulfilled by restricting the time a node can transmit to only after its predecessors have already transmitted. This mechanism is called ***precedence constraint***.

The aggregation convergecast problem can be simplified to the case of a single snapshot (data collection from all sensors) to be completely processed in a single round, *single-round constraint*. If the data collection must be completed in a single round, a reduction on the number of time slots has the effect of decreasing the collection latency and increasing the throughput.

However, the *single-round constraint* may be too restrictive for some applications. If the objective of an application is to have the highest collection throughput, independent of the collection latency, the requirement of having to complete a snapshot collection in a single round must be relaxed, and a form of ***pipelining*** should be adopted. *Pipelining* can be understood as a form of concurrency, where different nodes in a path toward the sink can transmit at the same time. *Pipelining* allows that a new data collection begins before the previous one has been finished, such that there are more than one data collection being executed at the same time.

A trivial, but essential requirement present in any feasible aggregation convergecast solution is that there must exist at least one path from each sensor node to the sink, otherwise the communication between some nodes and the sink would not be possible. We call this restriction ***reachability constraint***. An aggregation tree naturally satisfies this requirement. This requirement will be further discussed in Chapter 5.

Our thesis studies the use and interaction of these mechanisms: *non-conservative flows*, *resource constraints*, *precedence constraints*, *pipelining* and *reachability constraints*. These mechanisms affect the selection of the logical routing topology and the scheduling of transmissions in an aggregation convergecast solution. Let us formally define them.

1. **Non-conservative flows**

   The flow conservation property states that the total flow f out of a vertex u toward any vertex v other than the source s or sink t is zero (using the skew property [32])

$\sum_{u \in V} f(u, v) = 0, v \in V - \{s, t\}$. If the sum is not equal to zero, than we have a non-conservative flow. When the sum is negative, there is more data exiting then entering. If the sum is positive, there is more data entering the node than exiting.

2. **Resource constraints**

   Resource constraint is defined as a restriction where a resource has limited capacity to provide the service being requested. In our context, the use of wireless communication brings the possibility of different conflicts for the use of wireless media. These conflicts cause failures which are summarized by the term interference. From the point of view of communication (and our work), the most important failure is the inability to receive. It happens when a message is not possible to be decoded by the intended receiver.

3. **Precedence constraints**

   Precedence constraint is defined as a restriction where an event can only occur after another event or set of events have already taken place. Precedence constraint is also called non-circular if a given event cannot be a predecessor of itself. In our context, it means that a node only transmits after a precedence restriction has been satisfied.

4. **Pipelining**

   Pipelining is a form of parallelism that explores the possibility of concurrent transmissions among nodes belonging to the same aggregation path. If nodes on a path toward the sink is understood as serial stages in an assembly line, pipelining allows a node (stage) to be scheduled (executed) as soon as its transmission (execution) does not conflict with the aggregation (assembly) of the previous instance (product).

5. **Reachability constraints**

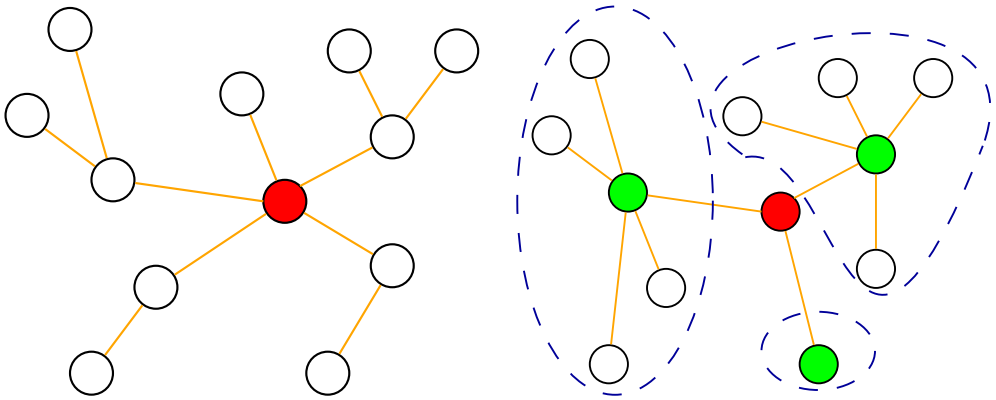
   For a directed graph $D = (V, A)$, the reachability relation of D is the transitive closure of its arc set A, which is to say the set of all ordered pairs (s, t) of vertices in V for which there exist vertices $\{v_0, v_1, \cdots, v_d\}$ such that $(v_{i-1}, v_i) \in A, \ \forall i \ 1 \leq i \leq d$, [96]. In our context, we reduce the transitive closure to the arc set A' of ordered pairs $(s, t = sink)$. We are only interested if each vertex is able to reach a single special vertex, called sink.

2.2 Solution Parts

The aggregation convergecast solution is composed of two parts: the routing part and the scheduling part. The routing part is expressed by a logical topology structure, or the nodes and links where the sensed information flows toward the sink. The scheduling part represents the selection of the time when each transmission event (communication from one node to another through a specific link) takes place, in order for the sensed information to reach its destination. Both parts are closely related.

2.2.1 Logical Topology

One crucial ingredient for a WSN application is a carefully selected logical topology. Depending on which is the objective: minimize the collection latency per round, or maximize the data collection throughput, a specific logical topology may be more appropriate. The connections of the logical topology have to be selected such that they favor the selected objective.

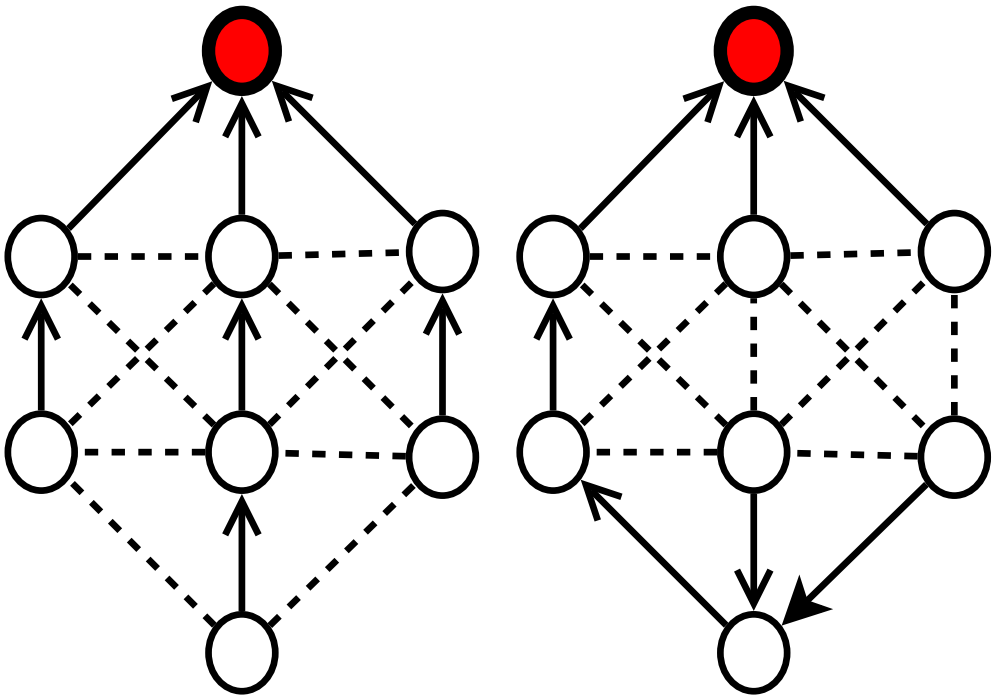


(a) Tree-Based (b) Cluster-Based

Figure 2.2: Hierarchical Topologies

Most studies propose the use of hierarchical structures. A tree-based approach, illustrated on Figure 2.2(a), is a classic strategy, based on a hierarchical organization of the nodes in the network. Tree-based topology is the simplest way to aggregate data flowing from the sources to the sink. It defines a preferred direction to be followed when aggregating data back to the sink. In this approach, a spanning tree, rooted at the sink, is constructed, then, it is used to transmit queries generated by the sink to the nodes and to transmit back responses from the nodes.

Trees can be constructed according to a variety of criteria. One of the most prevalent schemes used is Shortest Path Trees (SPTs), where the hop-count from a node to the sink is the minimum possible. It is exemplified in Figure 2.3(a). This approach is particularly suitable to design aggregation functions and to perform efficient energy management. Often, optimal paths are calculated in a centralized way at the sink, by exploiting different assumptions on the data correlation, and selecting the best aggregation points by means of cost functions. Several works use this scheme [6, 24, 38, 75, 76]. From the point of view of scheduling, the drawback of this scheme is that it only considers hop-count as the criterion for creating a logical topology (precedence constraints), without regarding interference effects on the scheduling (resource constraints). An alternative is the use of *non-SPT* topologies (Figure 2.3(b), where the number of hop-counts from some nodes to the sink are not minimal.



(a) SPT Topology (b) Non-SPT Topology

Figure 2.3: Tree construction criteria

The second common scheme is based on clusters, Figure 2.2(b). Cluster-based schemes also consist of a hierarchical organization of the nodes in the network. However, nodes are now subdivided into clusters. Special selected nodes, referred as cluster-heads (green nodes in Figure 2.2(b)), are elected in order to disseminate data locally and to transmit aggregated results back to the sink. Cluster-heads are frequently used to perform some

control function over the members of its cluster. DAS and PAS [112, 113] are examples of this scheme. Cluster-based topologies suffer from the same drawback of SPT-based topologies; and even worse, they create a bottleneck at the cluster-heads. The combination of precedence constraints and interference forces the serialization of transmissions from the cluster members to the cluster-head.

2.2.2 Schedule

Scheduling is the process of assigning a specific time to an activity to be executed. It can be periodic, meaning that after all activities are executed a new cycle starts again. In the wireless sensor network context, it consists of assigning a specific time for a node to transmit to another node.

In the absence of further requirement, a trivial schedule solution would be for one node to transmit after another, sequentially. This would result in a schedule length $n-1$. But WSN applications generally require another design objective such as throughput maximization, or network lifetime maximization. Such objectives preclude the use of this trivial solution.

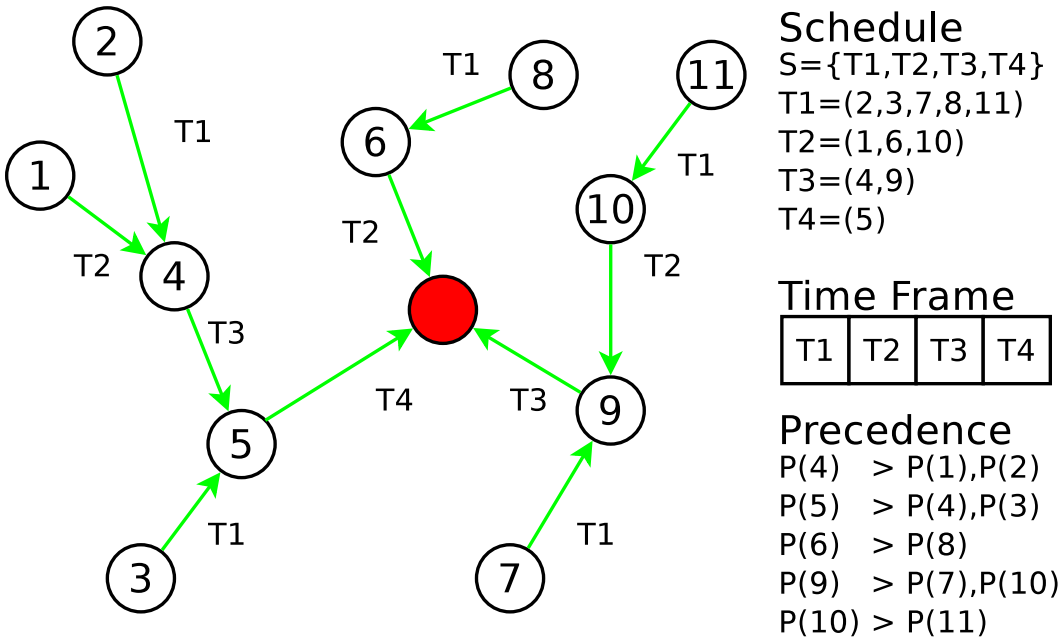


Figure 2.4: Aggregation Convergecast with Precedences in a Single-Round

Formally, a schedule can be defined as a set $S = T_1, T_2, ..., T_l$, where T_i represents the set of simultaneous transmissions executed on the time slot i, and l represents the *length*

of the frame (or cycle). A schedule is considerate *valid* with respect to some interference model if the model allows to schedule all transmissions from S without failure (i.e. without collisions) [107]. This problem is NP-Hard [84]. Figure 2.4 depicts an example of scheduling with *single-round constraints*, where all precedence constraints are satisfied in a single period. The design objective is to achieve the smallest schedule length. Figure 2.5 presents an example of pipelined scheduling, relaxing the *single-round constraints*.

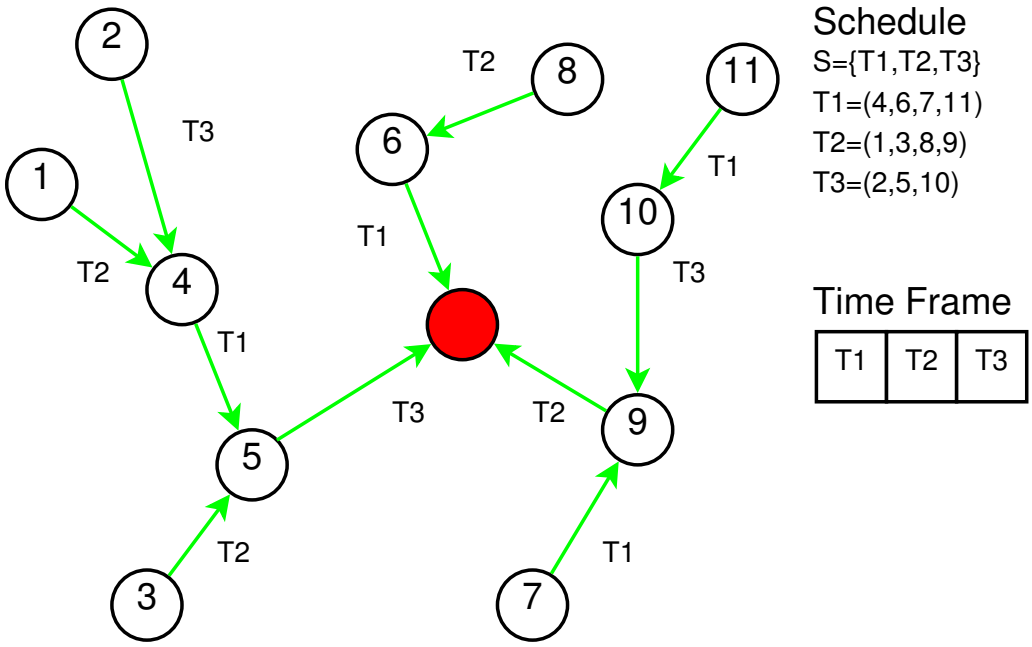


Figure 2.5: Pipelined Aggregation Convergecast

There are two common types of scheduling in the literature: *Link Scheduling* [8, 56] and *Node Scheduling* [40, 85, 109]. In link scheduling, the activation of a specific undirected link $\overline{uv}$ is the element used to define a transmission. In node scheduling, nodes are used to define the activation, and at least one of their neighbors receives the transmission.

Some wireless scheduling studies add additional restrictions. In the absence of a specific application to address, an abstract traffic demand is created, and the solution must fulfill this demand. The scheduling then varies according to the demand definition.

Some studies try to obtain the highest possible throughput, using **all** links/nodes available on the communication graph [14]. They do not consider any specific logical topology because all links are supposed to be used. They also do not have a specific control entity to receive all information collected.

Other studies create a specific traffic demand, generally expressed in terms of arbitrary source-destination pairs, or in terms of transmission rate, or in terms of amount of data to transmit. Then, they search for the minimal possible schedule length l [22, 63].

These described studies have a common trace: the absence of a combination of *resource constraints* (interference) and *precedence constraints*. In aggregation convergecast, these requirements are present and restrict possible scheduling solutions. If the precedence constraints are not present, the scheduling problem becomes a classical graph coloring problem [84], which has several heuristic solutions [41].

Another group of studies tries to obtain a solution based on a specific application and its restrictions. They search for a schedule which optimizes a specific design objective. The difference between this last group and the previous ones is the specificity of solution demanded by the application, instead of being a generic solution using all links and nodes or to accommodate a uncorrelated traffic demand. We approach the scheduling from the point of view of this last group.

2.3 Design Objectives

The data collection may have different objectives, depending on the application requirements. An application for the scenario presented in Chapter 1 can set the data collection to achieve different design objectives: one objective can be to minimize the data collection **latency**, and another design objective can be to maximize the data collection **throughput**. It is also possible to design the system to improve both objectives together in a multi-criteria optimization, or yet achieve first the best of a first objective, then try to optimize the next one. The relevance of the objectives used in our thesis are described next.

- **Latency Minimization**

  The minimization the data collection latency is relevant when an application is requested to take actions based on deadlines, such as mission-critical and event-based applications [59]. Minimization of latency may be attained by minimizing the schedule length, however, it may depend of other of characteristics of the input network, and also other restrictions.

- **Throughput Maximization**

  Throughput maximization can be an important objective for large, dense networks

and for applications that require efficient delivery of large amounts of data. This objective can be characterized by the rate at which collected information can be delivered to the sink [78].

The most used method to minimize the data collection latency and to maximize throughput is the minimization of the schedule length. Small schedule length translates into quicker data collection. If a complete data collection must be achieved in a single period, both objectives are optimized at the same time, otherwise, they are decoupled, and a smaller schedule length increases the throughput, but not necessarily the latency. The schedule length minimization is, by far, the most used method to achieve such objectives.

2.4 Models

2.4.1 Interference

The use of wireless communication brings the possibility of different types of radio communication failures [109]. The first type of failure occurs when a node attempts to send multiple messages at once, or it tries to decode multiple messages at the same time. From the point of view of communication, however, the most important failure is the inability to receive. It happens when a message is not sufficiently decoded by the intended receiver. Every transmission involves a sender node u and a receiver node v. If there is a concurrent transmission from a third node w, then the radio waves emitted by w interfere with the transmission from node u at receiver node v. The transmission quality is affected negatively. In the worst case, the receiver node v is unable to correctly interpret the transmission from node u. This is called interference. There are two widely accepted models to characterize interference in a wireless network, namely, the *physical model* and the *protocol model* [94].

The interference model is important to specify the relationship between the routing topology and the scheduling of the transmissions. The routing topology will influence/restrict the scheduling space (when a node can transmit) by the model of interference as well as the scheduling will influence/restrict the routing topologies (paths select to transmit information to the sink).

Figure 2.6 captures the essence of the model. Figure 2.6(a), 2.6(b), and 2.6(c) are the cases where there is some type of conflict. In Figure 2.6(a), the transceiver v can not transmit to transceiver w and receive from transceiver u at the same time. Figure 2.6(b) shows the case

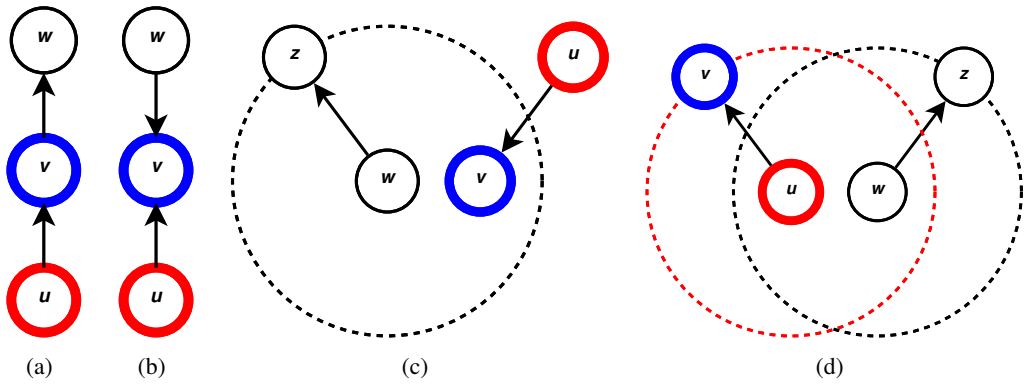


Figure 2.6: Interference Model

where two transceivers u and w try to execute a transmission to the same transceiver v at the same time, causing conflict. The transceiver v, in Figure 2.6(c), can not receive from transceiver u if transceiver w transmits at the same time to transceiver z, because it causes conflict. If in the three initial cases there is conflict, Figure 2.6(d) depicts a situation where, even though transceivers u and w are in transmission range, they can execute transmission concurrently, because the recipient of their respective transmission are out of range of each other.

2.4.2 Network and Application Models

Let a wireless sensor network application be defined as an undirected connected graph $G = (V, E)$, where V represents the set of nodes and E represents the set of edges. Let $|V| = n$ be the number of nodes on the network and $|E| = m$ be the number of links wirelessy connecting nodes in the network. Let i be the unique identifier of each node, and $xpos_i$ and $ypos_i$ its Cartesian coordinates. Each node i transmits with a power pwr_i. Let $s \in V$ be a central entity that coordinates all other nodes on the network. It is assumed that s has full knowledge of the network in terms of node positions, transmission power, time and application. The time is slotted.

Let $S = \{A(1), A(2), \cdots, A(r), \cdots, A(l)\}$ be the schedule such that $A(r)$ is the set of active arcs $\overrightarrow{uv} \in E$, where u represents the transmitter node and v represents the intended receiver node during the slot $r = \{1, \cdots, l\}$, where l represents the schedule length. Each complete execution of the schedule S accounts for one period p_j. The logical topology used is defined by $T = \bigcup_{r=1}^{l} A(r)$.

All nodes have precise knowledge about time, such that each node u transmits exactly on its allowed time slot r. There is no partial transmission or segmentation, such that every transmission from node u to node v begins and ends during the time slot r. Each time slot r has a duration of one time unit.

Each node i is composed of one transceiver capable of half-duplex communication. In simulations we adopt the (Unit Disk Graph) UDG model, where all nodes operate at a single frequency, at the same power level. The transmission range d_{tr} is fixed and equal for all nodes. When the distance between nodes u and v is $d_{uv} \leq d_{tr}$, node u can transmit and be received by v, and symmetrically, when node v transmits it will be received by u. The interference range is assumed to be equal to the transmission range. The only reason for a communication failure is due to interference. We assume *link activation model* [28] where interference only matters at the receiver node. The scheduling solution needs to be collision-free at each intended receiver, without need of carrier sense or other conflict resolution protocol. All nodes are static.

The time used for the execution of the in-network data processing operation is negligible. The aggregation is executed during the time slot when a node is receiving a transmission. At each period p_j, each node i generates a new sensor datum. A complete network-wide aggregation regularly is completed in one period p_j since the first period (non-pipelined), otherwise it is declared that a complete network-wide aggregation is completed using more than one period p_j (pipelined). Every node must transmit in one and only one time slot during a snapshot collection.

2.4.3 Optimization Problem

The formal optimization formulation for the aggregation convergecast problem has the parameters listed in Table 2.1. It follows the formal description of the problem presented in [76]. A optimization model for the pipelined aggregation convergecast problem is presented in Chapter 5.

$x^{t(v_i)}(v_i, v_j)$ is a boolean variable, which, if evaluated to 1, indicates that arc $\overrightarrow{v_i v_j}$ is active and a transmission from node v_i to node v_j takes place at time $t(v_i)$; if evaluated to 0, indicates that this arc is not active at time $t(v_i)$. The optimization problem is defined as follows.

Table 2.1: Aggregation Convergecast Parameters

n	Number of nodes
v_i	Node i
v_s	Sink node
$t(v_i)$	Transmission time of node v_i
$r(v_i)$	Recipient of transmission from node v_i
l	Maximum transmission time t (or schedule length)
$\Gamma_1(v_i)$	One-hop neighbour set of node v_i
$x^{t(v_i)}(v_i, v_j)$	Transmission from node v_i to node v_j at time $t(v_i)$

$$\text{minimize } l$$

such that:

$$\sum_{j \in \Gamma_1(v_i) \setminus \{v_s\}} \sum_{t=1}^{l} x^t_{(v_i,v_j)} = 1 \tag{2.1}$$

$$\sum_{j \in \Gamma_1(v_i) \setminus \{v_s\}} \sum_{t=t(v_i)}^{l} x^t_{(v_j,v_i)} = 0 \tag{2.2}$$

$$\sum_{j \in \Gamma_1(r(v_i))} x^{t(v_i)}_{(r(v_i),v_j)} = 0 \tag{2.3}$$

$$\sum_{j \in \Gamma_1(r(v_i)), v_j \neq v_i} \sum_{w \in \Gamma_1(v_j)} x^{t(v_i)}_{(v_j,v_w)} = 0 \tag{2.4}$$

Restriction 2.1 enforces a single transmission per node, except for the sink node. Restriction 2.2 ensures that, once a node transmits, it can no longer be the recipient of any transmission in the same period. Restriction 2.3 imposes that a node can not transmit and receive in the same time slot (half-duplex). Restriction 2.4 express the requirement that can be no interference at the receiver node. The problem formulated above has been proved to be hard by Chen *et al.*[24], even if restricted to Unit Disk Graphs (UDGs). Therefore, proposed solutions in the literature are heuristic approximations, or exhaustive search using some combinatorial tool.

Chapter 3

A Constraint Satisfaction Model for Aggregation Convergecast

3.1 Introduction

In this Chapter, we consider the modeling of aggregation convergecast problem in wireless sensor networks. We express the problem constraints in a manner compatible with a constraint programming solver, and extract solutions for small size instances.

Contrary to general purpose computing and networking systems, wireless sensor network systems are usually conceived with a specific application in mind. The application is typically the measurement and reporting of certain physical quantities. Little re-configuration takes place in the post-deployment phase. Generality and flexibility are sacrificed in the interest of reduced cost. This is to no small measure because of the limited node resources. In this spirit, wireless sensor networks usually form their own isolated and specialized network, allowing us to develop protocols tailored specifically to the application running on them.

There is the potential for several combination of paths to aggregate the data in the aggregation convergecast problem. The union of those paths can be represented by a spanning tree, which we call the *aggregation tree*. Determining the spanning tree and its corresponding optimal makespan schedule has been shown to be an NP-hard problem [24].

The existing approximation heuristics decompose the problem into two steps: a spanning tree construction, followed by scheduling, using some heuristic [24, 76, 112]. Shortest Path Trees has been the favorite aggregation tree. However, as of today, there has been no clear indication as to whether SPTs could result in an optimal schedule. The difficulty of course

for making any statement about the potential of SPTs is that the quality of the outcome of all proposed approximations is a function of *both* the aggregation tree *as well as* of the second *stage*, i.e., of the schedule construction heuristic (*given* an aggregation tree). In other words, it has been unclear whether the SPT alone or the subsequent scheduling phase is to be blamed when inefficient schedules are produced.

In this Chapter, we attempt to elucidate the tradeoffs behind the selection of the optimal tree. To this end, we do not restrict the family of aggregation trees in any way, and we do not perform any approximations, thus allowing us to observe if indeed the SPTs appear in better, or even optimal, schedules than what the current literature produces. The evidence suggests that the resultant aggregation trees is almost never SPTs and point to alternative considerations that would be useful when constructing aggregation convergecast scheduling heuristics.

This Chapter is conceived to present the aggregation convergecast problem in a standardized Constraint Satisfaction Problem (CSP) form that ensures no misunderstandings about the nature of the underlying optimization objective and constraints.

Section 3.2 presents an overview of Constrain Programming paradigm. It is followed by Section 3.3, where we provide an overview of aggregation convergecast trees. Section 3.4 explains what are the variables and constraints used in our aggregation convergecast CP model. Section 3.5 describes our findings using the model, and Section 3.6 provides the conclusions.

3.2 Constraint Programming Overview

Constraint programming (CP) is a paradigm for solving combinatorial problems that has roots on a wide range of ideas from artificial intelligence, computer science, databases, programming languages, and operations research. Constraint programming is currently used on many domains, such as scheduling, planning, vehicle routing, configuration, networks, and bioinformatics [89]. Constraint programming is a paradigm where the idea is to specify **what** the problem is by means of constraints rather than defining in detail the steps of **how** to calculate the solution. Constraints are just relations, and a constraint satisfaction problem states which relations should hold among the given decision variables. Once the problem has been modeled, the search for solutions is left to a constraint solver.

Constraint solvers take a CSP, represented in terms of decision variables and constraints, and find an assignment to all the variables that satisfies the constraints. An method is used that alternates search and inference. Inference consists of propagating the information contained in one constraint to related constraints. Such inference (usually called *constraint propagation*) is useful since it may reduce the search space. The search encompasses the search of the solution space using techniques like backtracking or branch-and-bound algorithms.

In the following, we give a short overview on some notation and definitions related to constraint programming. Most definitions follow the ones used in [33, 89].

3.2.1 CSP Elements

A constraint satisfaction problem consists of:

- a set of variables $X = \{x_1, \cdots, x_k\}$
- a finite domain set $D(x_i)$ of possible values for each x_i variable
- a finite constraints set restricting the values that the variables can simultaneously assume

The CSP elements allow the programmer to model a problem by creating variables, limiting them to a certain domain, and defining relations between two or more of them in the form of constraints.

Constraints

A constraint C on X is a subset of the Cartesian product of the domains of the variables in X. C is a subset of $D(x_1) \times \cdots \times D(x_k)$. The tuple $(d_1, \cdots, d_k) \in C$ is a solution of the problem. The domain $D(x_i)$ of a variable x_i is often a set of integers or an enumerated set of values. Another possibility are set variables whose values are sets. Each solution assigns the value d_i to the variable x_i, for all $1 \leq i \leq k$. We also say that the assignment of a solution satisfies C. If $C = \emptyset$, then there exists no assignment that satisfies C.

A solution to a CSP is an assignment of a value $d \in D(x)$ to each $x \in X$, such that all constraints are satisfied simultaneously. Solving a CSP is to decide whether a CSP has a solution or not, to find some solution of the CSP, or to enumerate all solutions. Often for

a CSP, it is not enough to just find a feasible solution, but to find the optimum solution in relation to a certain criteria. This leads us to the Constraint Optimization Problem (COP). A very relevant COP is Constraint-Based Scheduling (CBS), described in [12].

Constraint Propagation

The limitations created by the constraints are implemented as *propagators*. Constraint solvers, employ *propagation*, a process whereby values that do not hold for the constraints, are removed from being possible values of the corresponding variables. The objective of the propagation is to reduce the domain of the variables to either achieve a solved assignment or fail.

Every time that a variable x_i is assigned, or the domain $D(x_i)$ is modified, the propagators associated to the modified variable are executed. The propagator checks if, with the reduction of the domain of x_i, it is possible to prune the domains of other variables, meaning that values from other domains can be removed without changing the set of solutions. This process is called *filtering*. If a propagator is able to filter other domains, then this action may trigger the execution of other propagators or another execution of the same propagator. This process is repeated until the domains reach a stable state or one of the domains becomes empty.

In most cases, propagation alone is insufficient to solve a problem. Note that the result of the propagation could be to *fail*, i.e., when one or more of the variables have no values left in their domain. A problem is considered *solved* if each variable has exactly one value to choose from. However, a problem can be *distributable*, if it is neither solved nor failed, or, there are more than one possible values on the domain of the variables. To further reduce the domain of a distributable case, *branching* is used. Branching is a way to reduce the domains of the variables without losing accuracy by removing solutions. Branching takes a copy of the variables, domains and constraints and adds a new constraint to it.

Search

The search techniques to explore the search space are backtracking search and local search. A backtracking search performs a depth-search traversal of a search tree. A node forking represents alternative choices that are available to be examined to find a solution. The constraints are used to trim forked subtrees with no solution. Backtracking is necessary to

assure that, if a solution exists, it will be found. During the execution of the exploration process, values are assigned to variables according to the rules defined by the branchers. This process only deals with the choices created by the branchers. By using only choices designated by the branchers, a search strategy is generic and can be reused for other problems. The variables are assigned one after another. Filtering takes place after each assignment. If the domain of a variable becomes empty, a backtrack step is performed and the next search node is checked out. This is repeated until a value has been successfully assigned to each variable, or the domain of the first variable in the search tree is empty.

The most common tree traversal technique is depth-first search. One problem with the depth-first search is, if a bad assignment is used in the initial stages of the search, then we have to visit all nodes of the possibly huge subtree before the bad decision can be reverted. One alternative is to use of Limited Discrepancy Search (LDS) [58]. LDS systematically searches all paths that differ from the heuristic path in at most a small number of decision points, or discrepancies.

Optimization

Constraint optimization problems are solved using a *branch-and-bound* search. Initially, a depth-first search is performed to find a valid solution $(d_1, \cdots, d_k)$ to the underlying constraint satisfaction problem. The corresponding solution value $s = f(d_1, \cdots, d_k)$ of the objective function generates an upper bound for the solution. Now, a new constraint is added to the set of constraints to enforce the upper bound just found $f(x_1, \cdots, x_k) < z$. The current solution is invalidated, a backtracking step is performed, and the depth-first search continues with the additional constraint. If the new CSP is unsatisfiable, then $(d_1, \cdots, d_k)$ is the optimal solution with value of z. The lack of a new solution ends the process. If the new CSP has a solution, then we have a new upper bound and the process of adding a new upper bound is repeated, until an unsatisfiable CSP is reached. The unsatisfiability proves the optimality of the last valid solution. As the search for the optimal solution can be quite long, it is also possible to define some timeout for the whole process, and get the best solution that can be found within the given time.

3.2.2 Gecode: Generic Constraint Development Environment

The described process takes place usually using some specialized CP solver, a highly optimized software for exactly this purpose. Several commercial and open source constraint solvers, based on different programming languages, are available. Among the C++ based solvers are: the widely known ILOG solver, from IBM [69], and the open source *GE*neric *CO*nstraint *D*evelopment *E*nvironment, *Gecode* [92]. Gecode is the CP Solver used in this Chapter, because it is free, well documented, and open for extension of its functionality. Gecode is very portable, written in standard compliant C++ and runs on a wide range of hardware (32bit and 64bit) and operating systems (e.g., Unix/Linux, MacOS X, Windows). Extensive reference documentation is available. Considering the performance, Gecode seems to be very competitive even with commercial state-of-the-art solvers. Gecode can be easily extended with new propagators (as implementations of constraints), variable domains, branching strategies, and search engines. Gecode's flexibility owes primarily to the fact that it offers a variety of models for the formulation of constraint programs: as domain constraints, as relation constraints, as distinct constraints. In Gecode, constraint optimization problems are solved using branch-and-bound [31], or depth-first search. A non-exhaustive list of constraints provided by Gecode are presented in Table 3.1.

Table 3.1: Examples of Gecode constraints

Domain constraints	Constrain integer values and variable arrays to values from a given domain.
Relation constraints	Enforce relations between variables and between variables and integers values, like $(>, <, =, \neq)$.
Arithmetic constraints	Enforce arithmetic operations, like $min(x, y)$, $max(x)$, $sqr(x) = y$, $mod(y) = c$.
Linear constraints	Post linear constraints to a variable or array, like $\sum_{i=0}^{\lvert x\rvert=1} a_i \cdot x_i = c$.
Counting constraints	Count how often values are taken by an array.
Graph constraints	Enforce some graph property using some variable. Example: $circuit(x)$
Bin-packing constraints	Constrain how many items can be packed into bins.
Distinct constraints	Constrain that integer variables take pairwise distinct values.

3.2.3 Modeling and Search Space Reduction

Modeling is the process of transforming problem requirements into a CP solver acceptable and efficient format. This process can be described by a continuous improvement cycle, which tries repeatably to answer certain questions:

- *Which are the variables to represent the problem?* This is the most basic question,

which must be revised frequently, because often the first model is not the most efficient.

- *How to represent the constraints of the problem?* A initial ILP representation might help, but it can be improved using different propagators.
- *Is there any relationship between the variables?* Often, the problem particularities may provide relationships that are not evident at first.
- *What is the optimization criterion?* The criterion depends on the design objective selected.
- *Is the branch heuristic selected efficient for the problem?* Constraint programming framework has a flexible search strategy. Some branching heuristics may have different effects during the search.

Searching is the most time consuming part of finding a solution when using a CP model. After a correct and stable model is obtained, it is necessary to find ways to reduce the search space, and incorporate them into the model, or into the search strategy. Some intuitions are presented below in the form of questions:

- *Is there a more concise model representation, with less variables?* It is not unusual to realize that a variable can be eliminated because it can be obtained by the combination of others variables.
- *What are the variables' lower and upper bounds?* The variable domain may be elusive. Frequently, the first bound selected is too loose. Some restrictions, intrinsic to the problem addressed, can be applied. One way to obtain domain reduction is to use deterministic bounds derived from other works. If an optimal solution is required, it does not help to search in a solution space when it is known that better solutions exist.
- *What is the most efficient propagator for each constraint?* A constraint does not have a unique implementation. The selection of a propagator that better suits the problem reduces the search time.
- *Is it possible to reduce the solution space with new constraints?* In some cases, redundant constraints may prune the search space faster due the constraint propagation.
- *Is the sequence of variables to branch compatible with the problem?* The sequence

of the variables to branch affects your search efficiency.

- *Are there any equivalent solutions produced by the model?* Equivalent or symmetrical solutions waste time in the search process. Additional constraints may be added to eliminate redundancies.

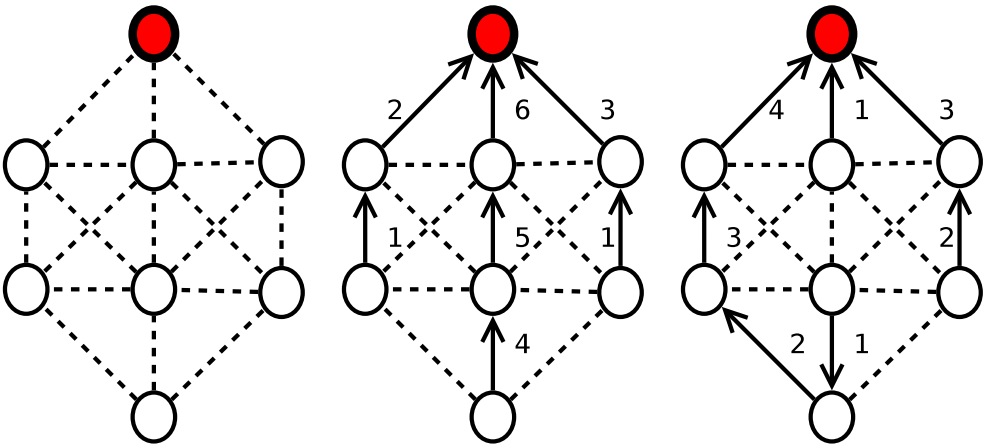
3.2.4 Constraint Programming in Networking

The main characteristics and techniques used in Constraint Programming are described in Section 3.2. The use of CP in networking related studies is not so rare. Recently, CP has been used to some degree to tackle network problems. In [74], CP is used to create a declarative constraint solving platform for policy-based channel selection and routing for multi-radio wireless mesh networks. It permits the specification of policy rules and operation constraints for channel selection and routing.

Voekler [107] studies the problem of scheduling with topology control in wireless networks. He considers networks with fixed transmission power, and networks with freely adjustable transmission power. He uses CP as a tool for solving scheduling problems subject to physical interference model. X-MAC [18] is a MAC protocol used for energy consumption reduction of sensor nodes by controlling the use of the radio transceiver. It is studied in [115] using CP to obtain appropriate parameters and to automatically adapt them to changes in network conditions and application requirements. Resource allocation problems in wireless mesh networks have also been approached as CP problems [22]. Simonis [95] presents a number of problems for network design, planning and analysis and show how they can be addressed with CP solutions.

3.3 Aggregation Convergecast Tree

One typical example of aggregation convergecast is presented on Figure 3.1. A base communication graph with the top node as the sink (where all sensed information should converge) is presented in Figure 3.1(a). Figure 3.1(b), shows a solution using SPT. The number next to the arrows represents the time slot when a link is allowed to transmit. This solution needs a time frame length of 6 time slots. Even for this small example, it is possible to obtain a better schedule, as presented on Figure 3.1(c), where the schedule length has only 4 time slots, but the solution uses a non-SPT topology as aggregation tree.



(a) Base Graph (b) SPT Schedule (c) Optimal Schedule

Figure 3.1: Schedule solution for example 8-node graph.

Previous works use SPT as the aggregation tree. In [24], the algorithm SDA follows the shortest data aggregation philosophy. It aggregates data along shortest paths towards the sink, incrementally constructing smaller and smaller shortest path trees. The algorithm DAS [112] constructs aggregation trees using connected dominating sets. The WIRES algorithm [76] uses a heuristic to obtain a variation of the SPT that produces better schedules, however it is still a SPT-based solution.

The number of the transmissions for a non-SPT topology will be the same as those of the SPT, because the aggregation convergecast problem already minimizes the number of transmissions (only one transmission will be performed by each node). However a SPT does not inherently account for the interference caused by node transmissions which, as we will see, impacts, as expected, the schedule length. So, it is preferable to select an aggregation tree that also minimizes the effects of the interference, and this tree is very likely not to be a SPT.

3.4 Variables and Constraints

The problem is modeled as an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges representing all communications links available in the transmission range of the nodes. The number of nodes in the graph is represented by n. A fixed node $s \in V$ is designated as $sink$. The graph G and the $sink$ node are the inputs for the CP solver. LSP represents the number of hops of the longest shortest path to the $sink$. If each

graph edge has weight 1, then LSP will represent the heaviest (across all nodes) shortest path from a node u to the sink. In this model, node u represents any node $\in V$ or the transmitter node on a communication link, while v represents the receiver node. $N(u)$ represents the (one-hop) neighbor set of node u.

Table 3.2: Model Variables

Variable	**Size**	**Domain**	**Meaning**
PP[u]	n	$1 \cdots n$	Parent of node u
RR[u]	n	$1 \cdots n$	Rank of node u
SH[u]	n	$1 \cdots n$	Schedule of node u
CC[t]	n	$1 \cdots n$	Parallel TXs on slot t
makespan	1	$1 \cdots n$	Optimization criteria

The model to represent the aggregation convergecast is composed of several variables, described in Table 3.2. The variable PP is defined for each node and represents the parent of the node in the definition of the tree. Each node must have only one parent. Each node transmits only once and only to its parent. The variable RR is defined for each node and represents the number of hops to the $sink$ using the selected tree. This variable is used to ensure that loops are avoided. The variable SH is defined for each node and represents the time slot in which the node transmits. The variable CC is defined for each time slot and represents the number of concurrent transmissions in the slot. CC is used to optimize the search process. The minimum time frame length is the final objective, so a minimization process is necessary, i.e., not just the generation of a feasible solution. The optimization criteria is represented by the variable $makespan$, which stands for the length of the TDMA schedule. It is trivial to observe that the value of SH variables need not be larger than n (the number of nodes) since this is the maximum number of time slots necessary when no concurrent transmission are possible. By the same token, the values of RR, PP and $makespan$ are upper bounded by number of nodes n.

$$PP[u] = v \;\Rightarrow\; RR[u] = RR[v] + 1 \tag{3.1}$$

$$PP[u] \neq v \begin{cases} u \rightarrow v \notin E \\ u \neq sink \end{cases} \tag{3.2}$$

The above constraints are the topology constraints and have the objective of limiting the communication graph, such that the final results is a spanning tree of the original graph, rooted at the $sink$ node. Constraint 3.1 specifies that the rank of the node u must be the

rank of its parent node v plus one. Constraint 3.2 restricts the nodes that can be parent of node u to only those which have a direct link to it.

$$PP[u] = v \;\Rightarrow\; SH[u] \neq SH[j] \begin{cases} j \in N(v) \backslash \{u\} \\ u \rightarrow v \in E \\ v \rightarrow j \in E \end{cases} \tag{3.3}$$

Constraint 3.3 captures the requirement that there can be no interference at the receiver node, based on the link activation model [28]. As before, the restriction is only valid when a specific link $u \rightarrow v$ is valid (i.e., if the edge exists).

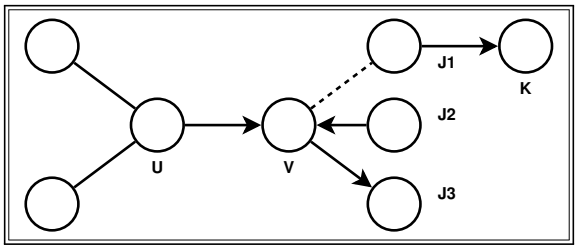


Figure 3.2: Types of interference

The Figure 3.2 shows the conflicts that are avoided between links that share the same neighbor using this constraint. The link transmission $u \rightarrow v$ conflicts with with link transmission $j_1 \rightarrow k$ because node j_1 is neighbor of node v and its transmission would jam the reception on node v, represented by the dashed line. The link transmission $j_2 \rightarrow v$ is not allowed because it not possible to node v receive two transmissions at the same time. The last conflict (representing the half duplex requirement), between link transmission $u \rightarrow v$ and link transmission $v \rightarrow j_3$, is not restricted by the interference constraint, but we do not introduce an additional constraint to handle it, since it is subsumed by an application constraint, Constraint 3.7. Constraint 3.7 states that if a node is parent (like node v) of another node (like node u), it should transmit after it (toward node j_3), i.e., in a later time slot.

$$\left.\begin{array}{l} RR[u] = 0 \\ PP[u] = u \end{array}\right\} for\ (u = sink) \tag{3.4}$$

$$SH[sink] > SH[u] \quad \{u \in V \backslash \{sink\}\} \tag{3.5}$$

$$SH[sink] \geqslant LSP \tag{3.6}$$

$$PP[u] = v \;\Rightarrow\; SH[u] < SH[v] \tag{3.7}$$

This set of constraints is related to the aggregation convergecast application. Constraints 3.4 and 3.5 define the unique role of the $sink$ node. It is the beginning of the rank count, parent of itself, and it must be scheduled (this is a pseudo-transmission to itself acting as the end delimiter of the schedule) after any other node on the tree. Constraint 3.6 restricts the domain of the schedule of the sink node, by ensuring that it cannot be smaller than the longest shortest path. Constraint 3.7 captures the requirement that a parent node can only transmit after it has received transmissions from all its children.

$$CC[t] \geq CC[t+1] \tag{3.8}$$

The above constraint deals with the fact that there are multiple possible feasible solutions for the same schedule length. The difference between them is not relevant to the optimization process, but searching all solutions wastes a great amount of computation.

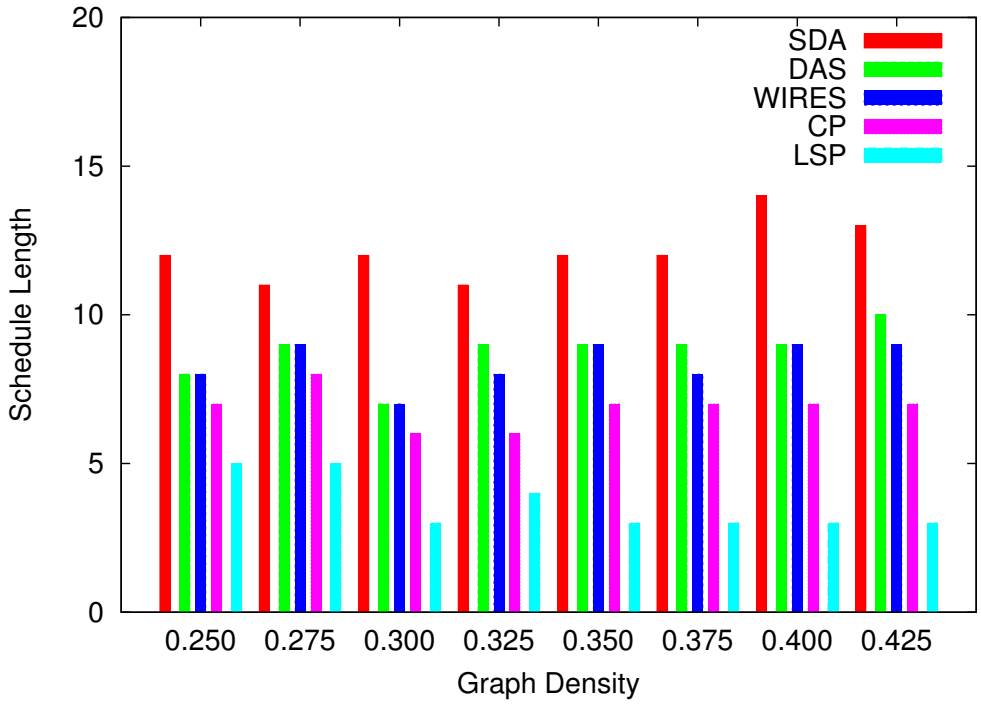
$$makespan = max(SH) \tag{3.9}$$

$$\textit{Branching:} \left\{ \begin{array}{c} PP[u] \\ SH[u] \end{array} \right\} \tag{3.10}$$

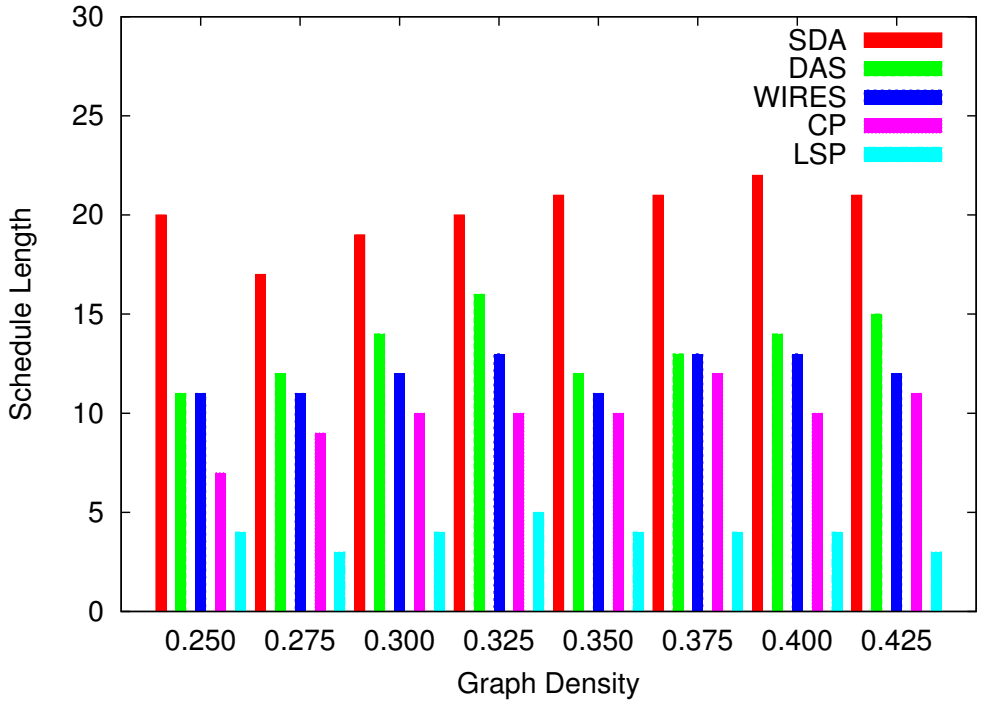
Constraint 3.9 defines how the schedule length is computed. Complementing the model, in Constraint 3.10, it is indicated which variables must be branched, and the order for such branching. The model only requires the branching of variable PP, which defines the tree to be selected, and variable SH, which defines the schedule for each node.

3.5 Results

Using the model presented in the previous section, it is possible to obtain solutions for the aggregation convergecast problem. Sample graphs were generated by randomly placing nodes on a square region with constant and uniform transmission range, forming Unit Disk Graphs (UDG). Groups of 10, 15, 20 and 25-node graphs were produced. On each group, the graph density ranges from 0.250 to 0.425 approximately. The graph density is defined

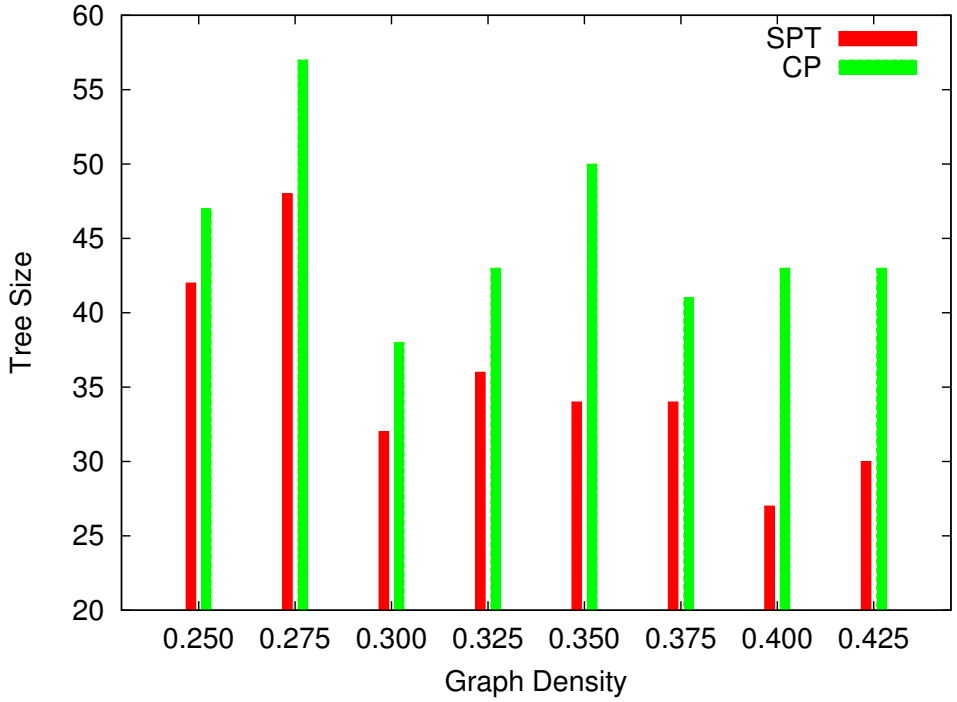


(a) 15-Node Graphs

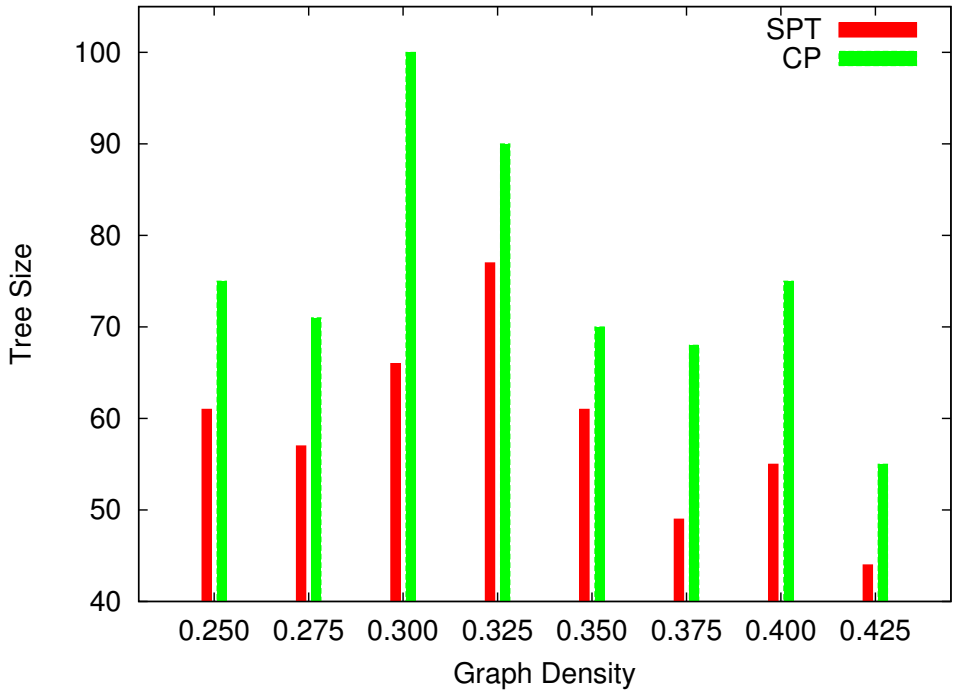


(b) 25-Node Graphs

Figure 3.3: Schedule Length Comparison



(a) 15-Node Graphs



(b) 25-Node Graphs

Figure 3.4: Tree Size Comparison

as the ratio of the number of links on the graph over the maximum possible number of links (for a completely connected graph with same number of nodes). Using graphs with different densities allows us to analyse scenarios with different levels of interference. Also, the aggregation tree size is defined as $\sum_{i=1}^{n} h_{(i,sink)}$, where h is the number of hops from node i to the $sink$ node using the selected tree. That is, the aggregation tree size is the sum of each node's hop-count to the sink along the aggregation tree. It is trivial to show that the smallest aggregation tree size for a graph is achieved by its SPT. A more complete description of the graphs used is presented on Table 3.3.

Table 3.3: Computational results from random graphs

Nodes	Density	Degree	Edges	WIRES	CP
10	0.2667	2.4000	12	5	5
10	0.2889	2.6000	13	5	5
10	0.3111	2.8000	14	5	5
10	0.3333	3.0000	15	6	5
10	0.3556	3.2000	16	6	5
10	0.3778	3.4000	17	6	5
10	0.4000	3.6000	18	6	5
10	0.4222	3.8000	19	5	4
15	0.2571	3.6000	27	8	7
15	0.2762	3.8667	29	9	8
15	0.2952	4.1333	31	7	6
15	0.3238	4.5333	34	8	6
15	0.3524	4.9333	37	9	7
15	0.3714	5.2000	39	8	7
15	0.4000	5.6000	42	9	7
15	0.4286	6.0000	45	9	7
20	0.2526	4.8000	48	9	8
20	0.2737	5.2000	52	8	7
20	0.3000	5.7000	57	10	7
20	0.3263	6.2000	62	9	7
20	0.3526	6.7000	67	11	7
20	0.3737	7.1000	71	11	8
20	0.4000	7.6000	76	13	9
20	0.4263	8.1000	81	11	11
25	0.2533	6.0800	76	11	7
25	0.2733	6.5000	82	11	9
25	0.3033	7.2800	91	12	10
25	0.3233	7.7600	97	13	10
25	0.3467	8.3200	104	11	10
25	0.3733	8.9600	112	13	12
25	0.4033	9.6800	121	13	10
25	0.4233	10.1600	127	12	11

Some of the existing (two-phase) aggregation convergecast scheduling heuristics are used for comparison purposes. The Balanced Shortest Path Tree (BSPT) [76] is used as the aggregation tree for all examples. For the second phase, three heuristic-based scheduling

Table 3.4: Minimal Tree Size for 15-node graphs

Density	WIRES	SPT Size	CP	Min TS
0.2571	8	42	7	43
0.2762	9	48	8	49
0.2952	7	32	6	34
0.3238	8	36	7	36
			6	41
0.3524	9	34	8	35
			7	36
0.3714	8	34	7	36
0.4000	9	27	8	28
			7	29

algorithms were selected: SDA [24], DAS [112] and WIRES [76].

CP solvers have the downside of spending a large amount of time to obtain an optimal solution, mainly in the search phase. As the number of constraints and candidate trees is related to the number of nodes and links, the time required for obtaining results can be prohibitive. Therefore, the CP results presented may not be optimal, but nonetheless they required fewer time slots than the heuristics produced.

The results obtained are depicted in Figure 3.3. Note that the results obtained using the CP model are closer to the minimal schedule possible (expressed by the LSP, used here to illustrate the lower bound for the convergecast scheduling problem) for the given graphs.

Figure 3.4 presents the tree size difference between BSPT, and the tree obtained from the best result using the CP model. It is noticeable that tree size obtained with the CP model is larger than the SPT, suggesting that it is not a good strategy to use SPT for this application. Table 3.4 depicts the minimum schedule length for 15-node graphs possible to obtain when we relax the tree size to be larger than the minimum possible size (SPT). Even though this conclusion seems intuitive, it is remarkable that the algorithms in the literature continue using SPT as base for their aggregation convergecast tree.

The difference between schedule length obtained by the previous aggregation convergecast algorithms compared to better solutions from the CP model is related to the characteristics of the tree selected for the aggregation path. Shortest path trees use the minimal number of hops from a node to the $sink$ node, independent of whether doing so might prove to be detrimental to the scheduling phase. This is typically the case when some bottleneck node is present on the graph, or specially when cliques are present. What we noticed is that SPT

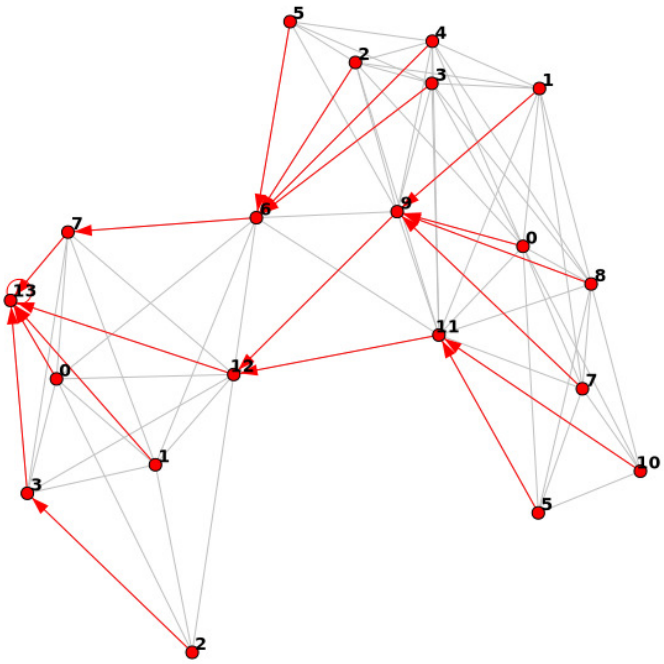
tends to use the clique internal edges to reach the nodes inside a clique because it produces, in most cases, the minimal path length to the $sink$. However, when the clique–internal edges are used, the nodes belonging to the same clique can not transmit in parallel. Even worse, if all clique–internal links are directed toward a single node (because it is the shortest path to the $sink$), the number of time slots cannot be reduced to less than the number of nodes in that clique. The final effect is that the overall number of time slots on the schedule cannot be smaller than the maximal clique on that graph.

Smaller schedules, found by the CP model, are possible by selecting links directed away from cliques (not using most of the clique internal edges) or bottleneck nodes (lots of incoming edges), increasing the size of the tree in order to avoid the serialization of the transmissions of nodes belonging to the same clique. This is the reason for the surprising relation between the results in Figure 3.3 and Figure 3.4. Whereas the tree can be significantly costlier (for example, the 25-node tree for 0.3 density presents the most pronounced difference) a shorter schedule can be achieved, even if the schedule is shorter by a couple of slots. One has to take into account that a couple slots difference might not appear as a small difference in the absolute sense, but relatively speaking it can represents a 10% (or even more) improvement over the schedule length produced by the best heuristic. Of course, due to the computation needs of the constraint solver, we cannot confirm if a similar proportional improvement is possible for larger networks.

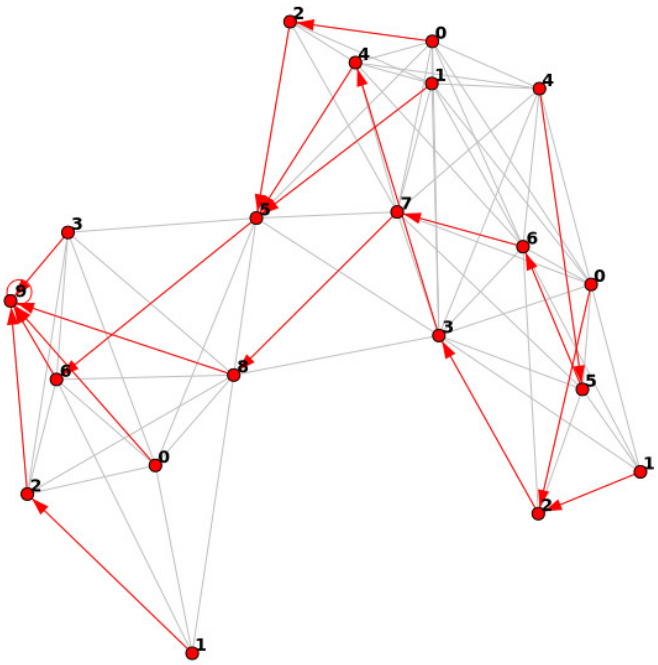
Figure 3.5 presents the solution achieved with the WIRES algorithm (on part (a)) of 13 time slots, and a solution obtained by the CP Model (on part (b)) of 9 time slots. The links with arrow indicate the aggregation path and the number beside the nodes shows the time slot when that node transmits to its parent. Notice how the clique in the upper right hand area is being partly *avoided* and how longer paths are effectively used reducing the in-degree of intermediate nodes in the aggregation tree.

3.6 Conclusions

We model aggregation convergecast scheduling as a Constraint Satisfaction Problem with the objective of obtaining the shortest possible schedule for a TDMA frame that allows the complete collection of data to the sink. Constraints related to the topology, the interference, and the application logic, as well as their relationships are modeled. We observe that SPT is very likely *not* the best choice for aggregation tree. While computational limitations



(a) Tree and Schedule using WIRES, Tree Size: 43, Schedule Length: 13



(b) A solution from CP Model, Tree Size: 57, Schedule Length: 9

Figure 3.5: Graph 20 Nodes - Density 0.400

do not allow us to derive results for large networks, the produced results provide valuable insights as to the **shape** that the optimal schedules and corresponding aggregation trees possess. Using non-SPT topologies that avoid the use of internal links to large cliques, has the potential to produce shorter schedules, improve the number of concurrent transmissions and hence the throughput of the solution. The advantage of topologies with longer paths suggests that other restrictions, beyond precedence, are important to obtain closer to the optimal solutions.

Chapter 4

Augmenting the Two-Phase Approach Using Convergecast Restrictions

4.1 Introduction

We explore in this Chapter the aggregation convergecast restrictions, their influence and how to use them for our benefit. A solution for the aggregation convergecast scheduling problem must satisfy the many-to-one aggregation process (expressed by precedence constraints), combined with the impact of the shared wireless medium (expressed by resource constraints). Both sets of constraints influence the routing and scheduling.

Several ways to aggregate the same data are possible, and each one of them may be thought of as represented by a different spanning tree, which we call an *aggregation tree*. Leaf nodes send their measurement to their parents. Interior nodes of the tree expect the results from all their children, and then perform the aggregation operation on the values coming from their children (and their own value) and finally send the aggregation result to their parent. In short, aggregation convergecast adds **precedence constraints** (parents transmit after receiving from all their children) to reduce overall traffic load. Additionally, due to the nature of wireless media, a node transmission also interferes with neighboring nodes who could transmit in the same time slot t. This restriction is termed **resource constraint**.

In the previous Chapter 3, we demonstrated that the selection of the aggregation spanning tree is important to reduce the schedule length and, while an SPT contributes to obtain a small schedule length by reducing the distance between the $sink$ node and all remaining

nodes on the network, it is unsuccessful in attaining the optimal schedule length. It is therefore conjectured that we need to search beyond the shortest path criterion to construct an aggregation tree. The inadequacy of SPTs is reinforced by the tree size metric (sum of the path lengths from all nodes to the sink) of the optimal solution obtained using the constraint satisfaction model of Chapter 3 which found the optimal aggregation tree sizes to be consistently larger than the tree size of the SPT. Under the light of the observations made in Chapter 3, all previous work in this area, that adopt SPTs, needs to be revisited.

Specifically, since Aggregation Convergecast Scheduling (ACS) can be viewed as subjected to two types of constraints (precedence constraints induced by the aggregation process, and resource constraints to express collision/interference avoidance), we can remark that SPT is a topology that reduces the impact on the schedule length caused by the precedence constraints (by reducing the number of hops from any node to the sink). Alone, this approach is inadequate for addressing the resource constraints. It is therefore instructive to balance precedence constraints and resource constraints. Towards this end, we will define the other extreme, i.e., a logical topology that captures the minimization of interference, and then propose a synthesis between this and SPT to construct trees that balance the impact from the two sets of constraints.

Additional to the nature of the spanning aggregation tree needed by ACS, it is necessary to address the scheduling part. Previous work on ACS complexity characterization has demonstrated that ACS is NP-hard [24]. Consequently, heuristics have been developed based on the decomposition of the problem in two steps: a spanning tree construction (usually SPT), followed by the slot-by-slot scheduling. As we will show in this Chapter, the scheduling step (given a spanning aggregation tree) is an NP-complete problem in its own right. In fact, it turns out that the scheduling step is a Mixed Graph Coloring (MGC) problem. Through the relation of ACS to MGC and other related varieties of scheduling problems, we will adopt corresponding solution strategies.

In this Chapter, we propose an aggregation tree construction suitable for aggregation convergecast that is a synthesis of a tree tailored to precedence constraints and another tree tailored to resource constraints. Additionally, we show that the scheduling component can be modeled as a mixed graph coloring problem. Specifically, the extended conflict graph is introduced, and through it, a mapping from aggregation convergecast to mixed graphs is described. In the mixed graph, arcs represent the precedence constraints and edges represent the resource constraints. The mixed graph chromatic number corresponds to the opti-

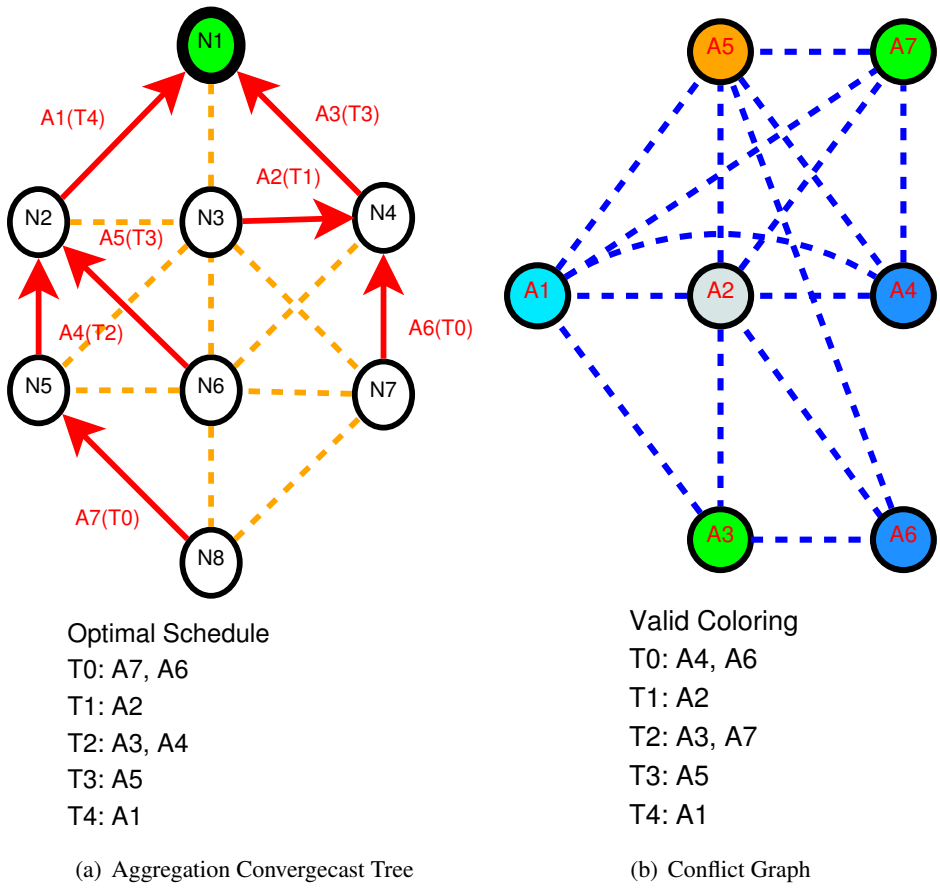
mal schedule length. Bounds for the graph coloring are provided and a branch-and-bound strategy is subsequently developed from which we derive numerical results that allow a comparison against the current state-of-the-art heuristic.

The rest of this Chapter is organized as follows. In Section 4.2 we introduce concepts that will be used throughout the Chapter. In Section 4.3, we present how to obtain an interference-aware logical topology, and how to blend it with another tree that minimizes the precedence constraints to form a single convergecast tree. In Section 4.4 we show how an aggregation convergecast scheduling problem can be represented as a MGC problem, establishing that ACS is NP-complete given a spanning aggregation tree and leading us to a branch-and-bound algorithm to search for the minimal MGC solution. By combining the results about the construction of the convergecast tree and the branch-and-bound algorithm, a series of computational experiments and discussion of their results is presented in Section 4.5. The most relevant works in this area are summarized in Section 4.6. Finally, the conclusions of the Chapter are presented in Section 4.7.

4.2 Background Basics

We will frequently refer to the *conflict graph* view of the network. Essentially, a way to achieve a conflict-free schedule is using conflict graphs derived from the topology (communication) graphs [21]. Each vertex in the conflict graph represents a link in the communication graph, and each link in the conflict graph represents the conflict between two links of the communication graph, which cannot be scheduled successfully (i.e., without resulting in a collision) in the same time slot. The basic idea of the conflict graph representation is that every independent vertex set on the conflict graph can be scheduled simultaneously, i.e., in the same slot. Therefore, the coloring of a conflict graph defines a valid schedule [9].

However, a valid coloring of the conflict graph is insufficient for providing a valid solution to ACS. The reason is that the conflict graph does not capture the *precedence constraints* of ACS. To illustrate this limitation, consider Figure 4.1. Specifically, Figure 4.1(a) shows the optimal schedule solution and the corresponding aggregation tree for ACS on a given (physical topology) graph. The labels next to each arc indicate the arc's id and its respective transmission timeslot. In Figure 4.1(b) we show the conflict graph corresponding to the same physical topology as before, and a valid coloring of the conflict graph. The coloring



(a) Aggregation Convergecast Tree

(b) Conflict Graph

Figure 4.1: Use of a conflict graph to create a schedule for ACS

alone does not provide a specific sequence in which all similarly colored edges are to be activated. For example, suppose the (arbitrary) order is blue, gray, green, orange, light blue, then the sequence of transmissions is as depicted at the bottom of 4.1(b). This sequence violates the precedence constraints as it can be seen by the fact that transmission $A4$ is scheduled before transmission $A7$. To overcome the limitations of conflict graphs, we will extend the model, subsequently calling it the *extended conflict graph*, and we will treat it as a Mixed Graph Coloring (MGC) problem in Section 4.4. In the same section, we show that the aggregation convergecast scheduling part can be converted into a mixed graph coloring problem, which is a NP-Complete problem [87, 88]. With this model, it is possible to see that ACS has the same restrictions (precedence and resource constraints) as scheduling problems in other fields, such as Job-Shop Scheduling [101] and Multiprocessor Task Scheduling [15]. We use our understanding of the problem restrictions (precedence and resource constraints) to convey an enumeration process to search for the chromatic number of a mixed graph. This process finds progressively smaller schedule lengths for ACS.

4.3 Topology Selection

4.3.1 Interference-Aware Aggregation Trees

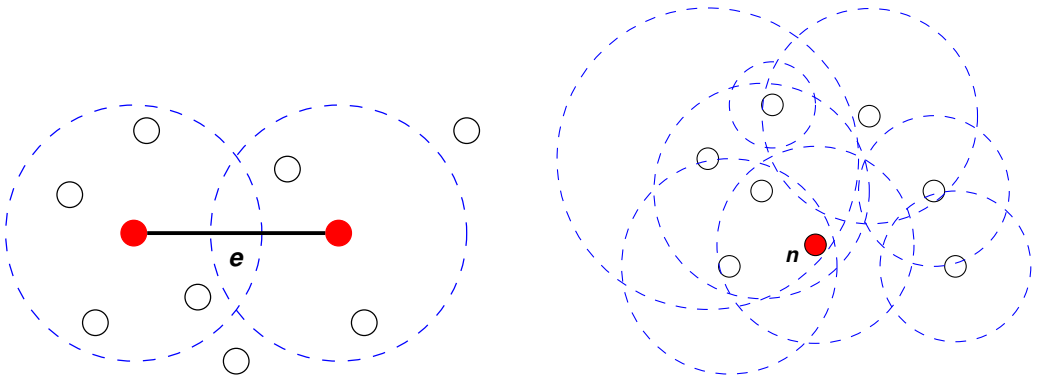
Clearly, interference is a component that affects the schedule length, because long schedule lengths may be necessary to accommodate transmissions without collision. We also note that there always exist a trivial feasible solution to the scheduling problem: one where one and only one node transmits in each time slot[1]. The question is if it is possible to obtain an aggregation tree that accounts for interference in its logical topology to reduce the effects of interference on the schedule. The answer has been partially explored in [109], which deals with the problem of finding *low-interference topologies*, topologies that minimize interference according to certain metric. There, two different interference metrics are presented: an *edge-based* metric and *node-based* metric. The *edge-based* metric counts all nodes that are within transmission range of either source or the destination nodes as the ones that may suffer interference. This interference metric is illustrated in Figure 4.2(a). A second way to represent interference is depicted on Figure 4.2(b), called *node-based*. This second metric counts from how many elements a node n receives interference.

Both metrics are limited, from the tree selection point of view, because neither takes into account the direction of flow. To this end, we adopt a new interference metric, more suitable to ACS. The new interference metric accounts for interference that would be caused if an arc $\overrightarrow{uv}$ is selected. This new interference metric is presented on Figure 4.2(c). Note that, under this definition, interference is not symmetric, because the interference caused by arc $\overrightarrow{uv}$ is not necessarily the same as the interference caused by arc $\overrightarrow{vu}$. Formally, we have:

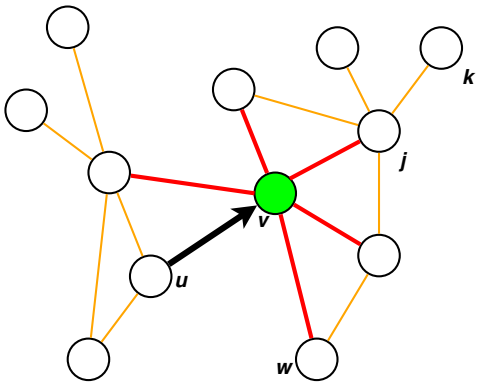
$$X_{arc}(\overrightarrow{uv}) = in\_degree(v) \tag{4.1}$$

The rationale behind this interference metric is that the arc direction matters. When a highly connected node receives a transmission, nodes in its vicinity can not transmit (neighbors are "blocked" in that time slot). However, when the same node transmits, some of its neighbors may also transmit. On Figure 4.2(c), if node v transmits to node w, node j can transmit to node k. This difference was not captured in other interference metrics, because they did not assume knowledge of the applications' communication needs (i.e., where to send a transmission next, that is, direction) but rather adopted a node-centered view. Furthermore,

[1]This solution happens to be the optimal in the special case of a completely connected communication graph. However, in general, this is not the case.



(a) $X_{edge}(e) = 8$ (b) $X_{node}(n) = 4$



(c) $X_{arc}(u\vec{v}) = 6$

Figure 4.2: Interference Measurements

the traditional algorithms (*Kruskal* and *Prim* [32]) to obtain a spanning tree T from graph $G = (V, E)$ where the total weight is minimized, use an undirected graph. The total tree weight is obtained by $w(T) = \sum_{(u,v)\in T} w(u, v)$. These algorithms are inadequate for constructing the minimum interference tree (MIT) because the edge weight varies according to the direction selected.

Consider a directed graph $G = (V, A)$, where V is the set of nodes and A is the set of arcs. Associated with each arc $\overrightarrow{uv}$ in A is a cost $c(\overrightarrow{uv})$. Let $|V| = n$ and $|A| = m$. The problem is to find a rooted directed spanning tree, $T_D(V, S)$ where $S \subseteq A$ such that $\sum c(\overrightarrow{uv})$ for $\forall \overrightarrow{uv} \in S$ is minimized. The rooted directed spanning tree is defined as a graph which connects, without cycles, all nodes with $n-1$ arcs (each node except the root) and has one and only one incoming arc. This formulation places the MIT construction in the class of *branching* problems, also known as *minimum cost arborescence* [10, 50]. An algorithm for solving this problem has been achieved independently by Chu and Liu [30] and Edmonds [39], while Karp [61] provided a combinatorial optimality proof. An efficient

implementation has been developed by Gabow *et al.*[46]. It is polynomially solvable [10].

Specifically, convergecast is an *in-branching* problem [10]. Therefore, a slight modification on Edmonds' algorithm is enough to obtain a MIT. The modification consists of changing the interference measurement used as weight on each arc to be the in-degree of the arc source. Upon termination of the algorithm, the resulting tree will be the transpose of MIT. The algorithm pseudocode is presented in Algorithm 1.

Algorithm 1: Minimum Interference Tree

Input: Undirected Connected Graph $G = (V, E)$, and source node s
Output: Minimum Interference Tree T_{MIT}
STEP 1: $G_{EDM} \longleftarrow$ create a DAG such that:
    (a) Transform each edge $e \in E$ into two arcs, one in each direction
    (b) Remove sink's incident arcs
    (c) Arc weight $w_{uv} = X_{arc}(\overrightarrow{vu})$
STEP 2: $T_{EDM} \longleftarrow$ Edmonds(G_{EDM}, s)
STEP 3: $T_{MIT} \longleftarrow T^{T}_{EDM}$
STEP 4: Return T_{MIT}

Complexity Analysis

We note that, with respect to the run-time complexity of *Minimum Interference Tree*, **STEP 1** part (a) is executed in $\mathcal{O}(|E|)$ because each edge should be changed; for part (b) the worst case is $\mathcal{O}(|E|)$ when the initial graph is full mesh; and for part (c), each edge must have its weight evaluated, which requires run-time of $\mathcal{O}(|E|)$. Therefore, the run-time complexity of **STEP 1** is $\mathcal{O}(|E|)$.

The analysis of **STEP 2** involves the run-time complexity of Edmonds' algorithm. According to Tofigh [104], Tarjan described an implementation of Edmonds' algorithm in [102] that runs in $\mathcal{O}(|E|log|V|)$. With a simple modification, the algorithm can run in $\mathcal{O}(|V|^2)$, which is more suitable for dense graphs. An implementation error is corrected by Camerini *et al.*in [20]. Gabow *et al.*[46] give an $\mathcal{O}(|V|log|V| + |E|)$ implementation for optimum spanning arborescence. The authors of [46] note that is not possible to improve on the time complexity for any Edmonds' algorithm implementation because the algorithm can also be used to sort n numbers, and sorting n numbers requires $\mathcal{O}(n \log n)$ time. Since it always has to inspect every edge of the graph, we cannot expect to find a better run-time complexity for Edmonds' algorithm than $\mathcal{O}(|V|log|V| + |E|)$. Therefore, we will assume that the run-time complexity for Edmonds' algorithm is $\mathcal{O}(|V|log|V| + |E|)$.

Finally, **STEP 3** is a trivial operation running in $\mathcal{O}(E)$. We conclude that the *Minimum Interference Tree* has a run-time complexity which is dominated by the execution of Edmonds' algorithm, i.e., $\mathcal{O}(|V|log|V| + |E|)$.

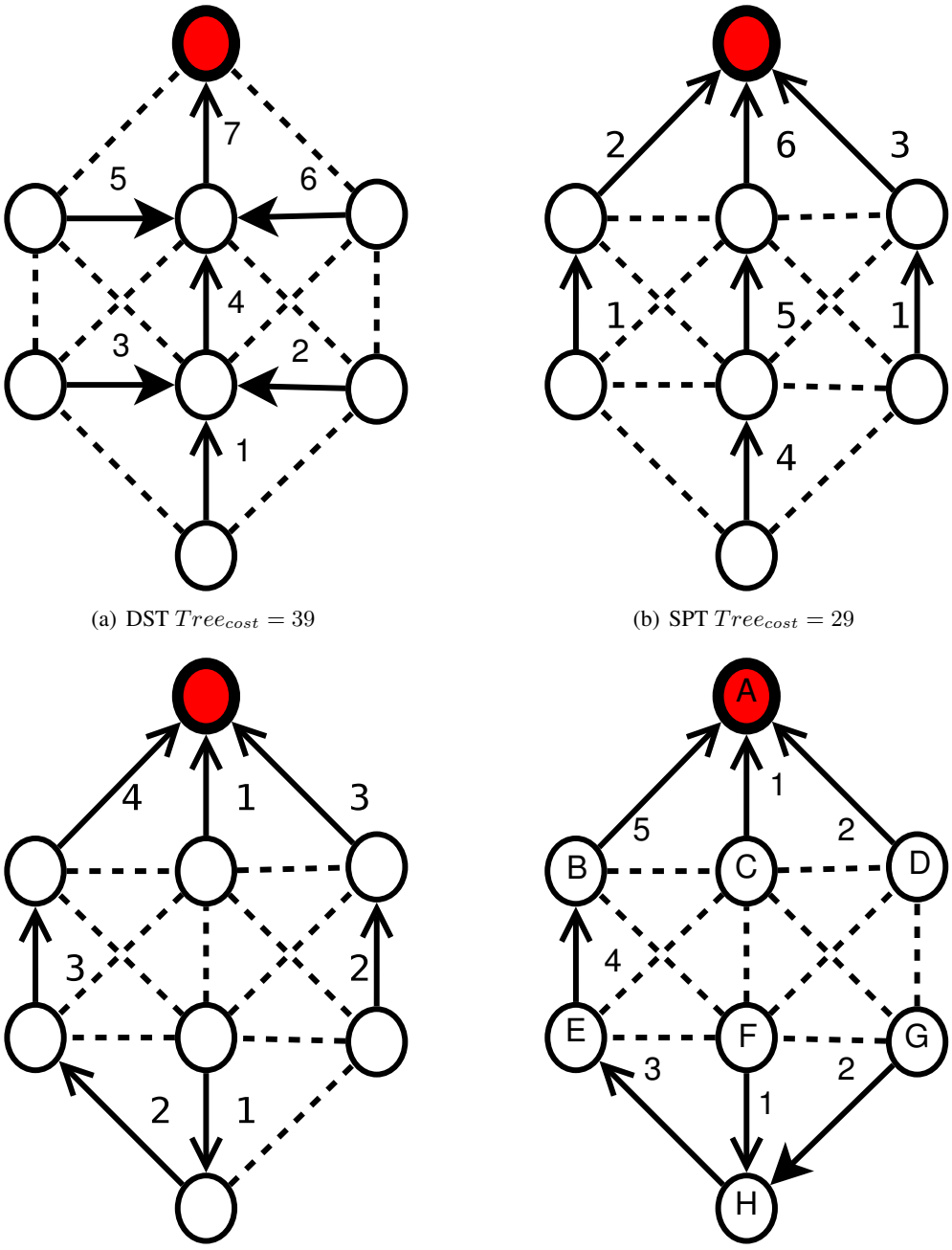
4.3.2 Trees for Combined Interference and Precedence Constraints

Using the interference metric of Definition 4.1, we calculate the total tree interference cost $c(T) = \sum c(\overrightarrow{uv})$ of each tree presented on Figure 4.3. Each graph represents a different logical topology organization. Figure 4.3(a) is a dominating set tree, Figure 4.3(b) represents a shortest path tree, while Figure 4.3(d) is a minimum interference tree. Finally, for reference, Figure 4.3(c) is an optimal tree obtained through a constraint satisfaction model. The tree total interference cost, according to X_{arc} metric, is presented below each graph. The schedule length obtained by each logical topology follows nicely the tree total interference cost, nevertheless MIT (on Figure 4.3(d)) is not the logical topology with minimum schedule length.

Just by looking at the provided example, it should be clear that even if the total interference cost is minimal, the schedule is not the shortest possible. A careful observation of Figure 4.3(d) shows that the long path from node F (or G) to node A is the cause of this longer schedule length. The fact that a MIT (or minimization of resource constraint) alone does not allow optimal solution is consistent with what happens when a SPT (or minimization of precedence constraints) is used alone. The logical conclusion is that both constraints must be addressed in a balanced manner, leading us to logical topologies that express a combination of the characteristics of SPT (Figure 4.3(b)) and MIT (Figure 4.3(d)).

The search for a logical topology that has both, approximately the minimum cost and approximately the shortest paths from all nodes u to a root node is not new in communication or circuit design [2, 27]. However, to the best of our knowledge, it has not been applied to scheduling problems to reduce the schedule length, using an interference metric as cost.

One approach to obtaining a suitable logical topology is the construction of Light Approximate Shortest-Path Trees (LASTs) from a given directed weighted graph, using the algorithm proposed by Khuller [62]. LASTs approximate simultaneously the cost of a minimum spanning tree (MST) and the distances of a SPT rooted at a source node, thus yielding a tree with low total cost as well as a short distance to the source node. This kind of tree is also called *Shallow Light Tree* (SLT). An algorithm to obtain such tree was initially proposed



(a) DST $Tree_{cost} = 39$

(b) SPT $Tree_{cost} = 29$

(c) Optimal $Tree_{cost} = 24$

(d) MIT $Tree_{cost} = 21$

Figure 4.3: Tree cost using X_{arc} metric.

by Awerbuch *et al.*[7]. SLT minimizes simultaneously both weight and depth, combined by means of a parameter.

Some modifications of the SLT algorithm are necessary to apply it to our context. One modification is the addition of **STEP 7** (of Algorithm 2), because we need that the arcs

converge to the root (sink node). Another modification is regarding the α parameter. In ACS, the important metric is the schedule length, which is outside the scope of the tree construction algorithm. It is therefore assumed that the selection of α is performed for an indirect purpose, i.e., based on the impact the constructed topology has on the subsequently constructed schedule length (detailed in the next section). The α parameter forces, for a given node v, to select the path $P^{MIT}_{v \rightsquigarrow s}$ (i.e. path over the MIT) if the weight of this path is not α-times the weight of the path $P^{SPT}_{v \rightsquigarrow s}$ (i.e. the corresponding path over the SPT). In the context of our problem, it means that a path with smaller sequence of precedence is preferable if the interference weight count of the path defined by the minimum interference tree is more than α-times the interference weight count of the path using smaller sequence of precedences. An algorithm sketch containing all the steps, from the initial undirected graph to the final aggregation tree, is presented next, in Algorithm 2.

Algorithm 2: Aggregation Convergecast Tree

Input: Undirected Connected Graph $G = (V, E)$, and source node s
Output: Aggregation Convergecast Tree T_{ACT}

STEP 1: Create a directed graph G^D with arc cost $c_{uv} = X_{arc}(\overrightarrow{uv})$
STEP 2: Find the min cost arborescence T_{MIT} from (G^D, s, c)
STEP 3: Find the shortest path tree T_{SPT} from (G^D, s)
STEP 4: Find a preorder sequence of T_{MIT}, using s as start node
STEP 5:
begin
    $H \longleftarrow T_{MIT}$;
    $\alpha \longleftarrow$ parameter that controls the SPT vs. MIT tradeoff;
    foreach *node v in the preorder sequence of T_{MIT}* **do**
        find a shortest-path $P^{H}_{s \rightsquigarrow v}$ in H ;
        find a shortest-path $P^{G^D}_{s \rightsquigarrow v}$ in T_{SPT} ;
        if $c(P^{H}_{s \rightsquigarrow v}) > \alpha \cdot c(P^{G^D}_{s \rightsquigarrow v})$ **then**
            add all arcs in the path $P^{G^D}_{s \rightsquigarrow v}$ to H ;
        end
    end
end
STEP 6: Find the shortest-path tree T_{SLT} of H with root node s ;
STEP 7: $T_{ACT} \longleftarrow T^{T}_{SLT}$;
STEP 8: Return T_{ACT}

Complexity Analysis

The run-time complexity of *Aggregation Convergecast Tree* is as follows: **STEP 1** has run-time complexity of $\mathcal{O}(|E|)$. Edmonds' algorithm is used in **STEP 2**, and has run-time

complexity of $\mathcal{O}(|V|log|V| + |E|)$.

STEP 3 has to determine a shortest path tree in a directed graph. Its complexity is dependent on the algorithm used to accomplish this task. The Bellman-Ford algorithm has a run-time complexity of $\mathcal{O}(|V||E|)$ [32], while Dijkstra's algorithm depends on how the priority queue is implemented [32]. We will assume a run-time complexity of $\mathcal{O}(|V|^2)$ [32]. The preorder sequence required on **STEP 4** is also of run-time complexity $\mathcal{O}(|V|^2)$.

The dominant run-time complexity is **STEP 5**. Lines 7 is linear to $|V|$, i.e., $\mathcal{O}(|V|)$, while Line 8 is constant $\mathcal{O}(1)$. Lines 10 and 11 both exhibit the same run-time complexity of $\mathcal{O}(|V|^2)$ because of the shortest path tree algorithm necessary. As they are inside the loop of Line 9, their resulting run-time complexity is $\mathcal{O}(|V|^3)$. Line 13 is not always executed, but when executed, it has a worst case of $\mathcal{O}(|V|^2)$. In total, **STEP 5** has a run-time complexity of $\mathcal{O}(|V|^3)$.

STEP 6 implements another shortest path tree algorithm with a run-time of $\mathcal{O}(|V|^2)$, and **STEP 7** has a run-time complexity of $\mathcal{O}(|V|)$. Therefore, the *Aggregation Convergecast Tree* algorithm has run-time complexity of $\mathcal{O}(|V|^3)$.

4.4 Scheduling Model

4.4.1 The Mixed Graph Coloring Problem

Let $G_M = (V, A, E)$ be a mixed graph, where $V = \{v_1, v_2, \cdots, v_n\}$ is a non-empty set of vertices. A represents the set of arcs, where (v_l, v_q) is an oriented arc from source l to destination q. E denotes the set of edges, where $[v_i, v_j]$ represents an edge connecting vertices v_i and v_j. $\mathbb{N}$ denotes the set of natural numbers. The number of vertices of G_M is $|V| = n$, where $n \in \mathbb{N}$.

The function $\varphi : V \rightarrow N$ is called *coloring* of the mixed graph G_M. The mixed graph coloring (MGC) problem assigns positive integers to vertices of a mixed graph such that, if two vertices v_i and v_j are linked by an edge $[v_i, v_j]$ then their colors have to be different $\varphi(i) \neq \varphi(j)$, and if two vertices v_l and v_q are linked by an arc (v_l, v_q), then the color of the start-vertex has to be smaller than the color of the end-vertex $\varphi(l) < \varphi(q)$. A *k-coloring* of a mixed graph G_M is a function $\varphi : X \rightarrow \{1, 2, \cdots, k\}$ such that $[v_i, v_j] \in E$, $\varphi(i) \neq \varphi(j)$ and for $(v_l, v_q) \in A$, $\varphi(l) < \varphi(q)$. There exists a number of different φ functions. The smallest possible number k is called *chromatic number* of the mixed graph G_M, and it is

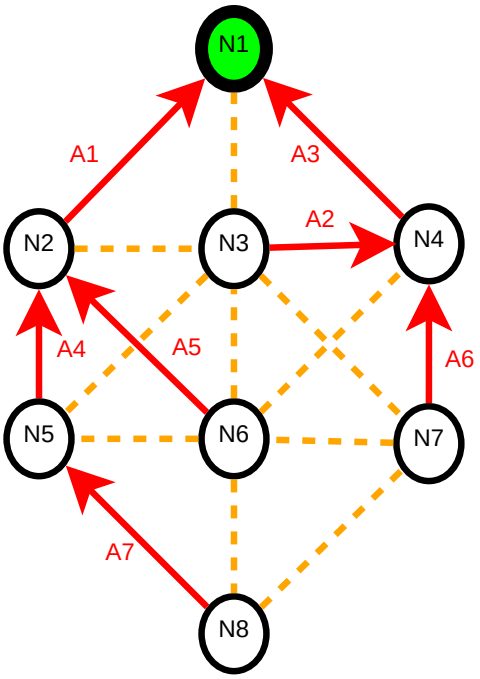
denoted by $\gamma(G_M)$. A coloring $\varphi : V \rightarrow \{1, 2, \cdots, \gamma(G_M)\}$ of the mixed graph G_M is optimal. A mixed graph G_M must be acyclic, otherwise no proper k-coloring is possible. The idea of mixed graph was introduced by Sotskov and Tanaev in 1976 [53].

4.4.2 ACS as a MGC

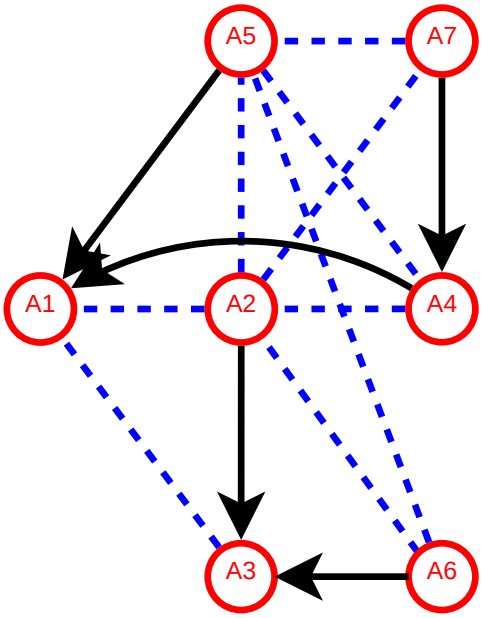
It is straightforward to notice the correspondence between ACS and MGC. The arcs of the mixed graph can represent the precedence constraints and the edges can represent the resource constraints among the transmissions. Additionally, a time slot in the schedule may be represented by a color assignment, which, in the case of an arc, can only increase, and in case of an edge, must be different. A precedence constraint corresponds to an arc in the mixed graph. The union over all arcs forms a tree rooted at a single vertex, the sink node s.

As remarked earlier, the coloring of a conflict graph [9, 21] is inadequate because conflict graphs lack the capacity to express the precedence constraints requirements of ACS. However, it is possible to represent ACS through mixed graphs derived from conflict graphs. Specifically, when a link activation (expressed by a node in the conflict graph) is required to be executed after another link activation (another node in the conflict graph), an arc is introduced in a mixed graph to express this precedence. This new *extended conflict graph* is the missing piece for a complete representation of ACS. In other words, the *extended conflict graph* is a mixed graph.

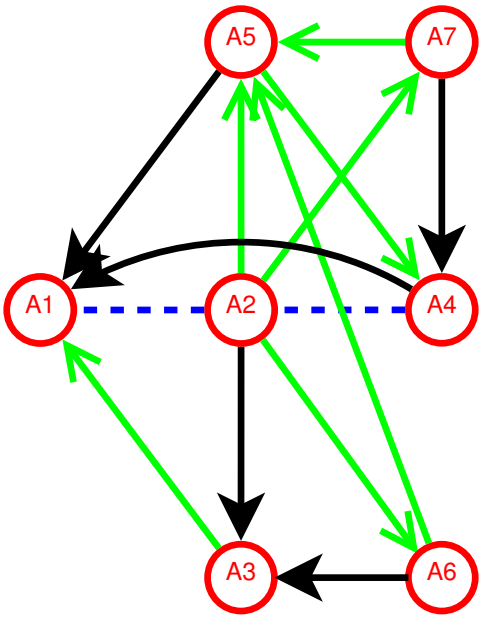
An example of reduction of the ACS problem to a mixed graph is presented in Figure 4.4. An aggregation convergecast tree is depicted in Figure 4.4(a). The link activations (arcs) of the aggregation tree are labeled from $A1$ to $A7$, and each time two nodes are in transmission range, the possible interference is represented by a dotted line. The same labels used to name link activations on the aggregation tree are applied on the vertices of the mixed graph, Figure 4.4(b). If a link activation can only be executed after another (e.g. $A4$ can only schedule after $A7$), this dependence is represented by an arc in the mixed graph. Dotted lines in the mixed graph represent a constraint between two activations. An immediate consequence of the mapping to MGC is that the ACS scheduling stage (that is, even if we are given the aggregation tree) is an NP-Complete problem [87, 88].



(a) Aggregation Convergecast Tree



(b) Mixed Graph Reduction or Extended Conflict Graph



(c) Mixed Graph after orienting all conflict edges

Figure 4.4: Reduction from AGS to MGC and Complete Edge Orientation.

4.4.3 ACS Bounds

Through its relation to MGC, we can provide some properties for ACS. The bounds presented here are applicable to the *Strong Mixed Graph Coloring Problem* [88]. A coloring

exists if and only if the mixed graph G_M does not contain any directed circuit (which is a requirement trivially satisfied for ACS instances given that its logical topology is a tree). The absence of directed circuits is a necessary and sufficient condition for a mixed graph to admit a strong mixed coloring.

Let P be a directed path in G_M. $|P|$ is the number of vertices on this path, and $n(G_M)$ the number of vertices on the longest path P on graph G_M. Let v_i be a vertex in the mixed graph. We denote $in(v_i)$ as the *inrank* of v_i, which is the length of the longest directed path in G_M **ending** in v_i. Likewise, $out(v_i)$ is the length of the longest directed path in G_M **starting** in v_i. This is the *outrank* of v_i. Also, the number of arcs incident to a vertex $v_i \in G_M$ is represented by $\Delta(G_M(v_i))$. Next, we present some propositions applicable to the aggregation convergecast problem.

Proposition 1. *Let $l(G_M)$ be the number of vertices on one of the longest directed path in G_M and $\Delta(G_M)$ the maximum arc in-degree of a vertex in G_M. Then, $\gamma(G_M) \geq max\{l(G_M), \Delta(G_M)\}$*

Proof. Different colors must be assigned to each vertex on the same directed path, consequently $\gamma(G_M) \geq l(G_M)$. In addition, all arcs incident on the same vertex must get different colors, otherwise a node could receive transmissions at the same time. Therefore $\gamma(G_M) \geq \Delta(G_M)$. Thus, $\gamma(G_M) \geq max\{l(G_M), \Delta(G_M)\}$. □

Proposition 2. *Let $G_M^0 = (V, A, \emptyset)$ be a partial graph from G_M. If $\gamma(G_M^0)$ is the chromatic number of G_M^0, or the length of one of the longest directed path in $\gamma(G_M^0)$, then $\gamma(G_M) \geq \gamma(G_M^0)$. $\gamma(G_M^0)$ is the lower bound and can be determined in polynomial time.*

Proof. This proposition is a direct consequence of complexity results from Ries [87] and Garey&Jonhson [49]. □

Proposition 3. *The chromatic number $\gamma(G_M)$ is equal or smaller to the cardinality of set V. In other words, $|V|$ is the upper bound of $\gamma(G_M)$.*

Proof. From [57] it is known that $\gamma(G_M) \leq \gamma(G_M^c) + |V| - \gamma(G_M^o)$, where G_M^c is the mixed graph without orientations, and G_M^o is the subgraph generated by the set V_o consisting of the vertices v_i, which have at least one incident arc. However, $\forall v_i, v_i \in V$, therefore $V = V_o$, and consequently $\gamma(G_M^c) = \gamma(G_M^o)$. Thus, $\gamma(G_M) \leq |V|$. □

4.4.4 Obtaining the Chromatic Number

Let us define the set $Q = \{1, \cdots, n\}$ as the set of transmissions of the aggregation convergecast problem. In Figure 4.4(a) it would have been represented by a set of vertices. The set of transmissions is partially ordered because of the precedence constraints of the problem. The arcs in the extended conflict graph represent this partial order of the transmission set. However, the partial order does not completely define a transmission schedule. The definition is possible only after the precedence between vertices (or transmissions) that are in conflict is resolved. That is, after we assign direction (converting them to arcs) to edges in the mixed graph. Three possibilities exist for each edge. Namely, given an edge represented by two vertices $[i, j] \in E$, either vertex i is scheduled before vertex j, or vertex j is scheduled before vertex i, or the orientation of this edge is irrelevant because the order was already established by the arcs (precedence constraints). The selection among the three possibilities will be determined through an objective function (detailed later), which we will call F. Therefore, the problem now is to find a feasible schedule to minimize the value of the objective function F, by orienting the edges that admit orientation in the mixed graph. For instance, the choice of the arc (i, j) defines the precedence of transmission i over transmission j. An example of this process is presented on Figure 4.4(c), where the *green arcs* represent the edges that were oriented, and the dotted blue lines represent edges not oriented, because their orientation is irrelevant to the schedule.

Theorem 1. *Let $P(G_M)$ be the set of all digraphs, created by orienting each edge of the mixed graph G_M. The digraph $G_s \in P(G)$ defines a feasible schedule if and only if G_s has no circuits.*

Proof. This theorem is straightforward to derive because if a digraph has a circuit, it does not define a feasible schedule. See also [101]. □

Let $\mathring{P}(G)$ be the set of all digraphs without circuits taken from the set $P(G)$. Each digraph $G_s \in \mathring{P}(G)$ defines a unique valid schedule of node transmissions. The task of obtaining the chromatic number is now to select a digraph G_s whose objective function $F(G_s)$ (the chromatic number of G_s) attains the minimum value among all feasible digraphs $G_s \in \mathring{P}(G)$. The cardinality of the set $\mathring{P}(G)$ has an upper bound of $\lambda(\mathring{P}(G)) \leq 2^{|E|}$, that is the maximum number of combinations of edge orientations.

In essence, the process of obtaining a valid schedule from a mixed graph consists of remov-

ing each relevant edge $[i,j] \in G(M)$ and replacing it by the arc (i,j) or the arc (j,i). The sequence of operations defines a sequence of transformations applied on edges. The proof that such sequence of transformations generates a valid digraph is presented in [97].

Theorem 2. *The digraph $G_s \in P(G)$ defines a feasible schedule for ACS if and only if two conditions hold:*

1. *It is the result of a sequence of transformations applied to the original extended conflict graph;*
2. *The digraph G_s on the extended conflict graph contains no circuits.*

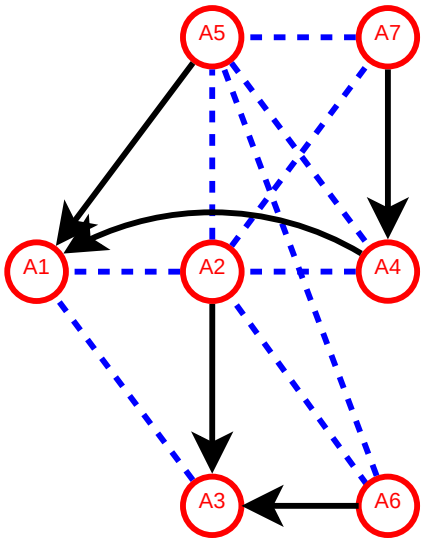
Proof. As we have seen before, an extended conflict graph represents the constraints of the aggregation convergecast scheduling problem, and G_s will be a valid schedule after the relevant edges have been oriented. □

4.4.5 A Branch-and-bound Algorithm

Next we define a way to select or search for a digraph $G_s \in \mathring{P}(G)$ that minimizes the objective function F noting that a sequence of transformations $T^1, T^2, \cdots, T^\omega$ defines an enumeration of all possible digraphs $G_s \in P(G)$. Each transformation consists of removing an edge and introducing an arc on the mixed graph, such that, each transformation outputs a different mixed graph G_M^k, where $k = \{1, 2, \cdots, \omega\}$. $l(G_M^k)$ may be different for each k, as well as its chromatic number $\gamma(G_M^k)$.

To this end we wish to obtain a sequence of transformations such that a digraph G_s that minimizes F can be found as fast as possible and, additionally, we avoid transformations that could create a mixed graph G_M^k with cycles. We will define conflict edges sets in *increasing level of conflict*, following the substance of the method outlined by Andreev *et al.*[5]:

- **Late conflict set (LC)**: $[v_i, v_j] \in LC$ if $out(v_i) = out(v_j)$
- **Early conflict set (EC)**: $[v_i, v_j] \in EC$ if $in(v_i) = in(v_j)$
- **Both conflict set (BC)**: $[v_i, v_j] \in BC$ if $out(v_i) = out(v_j)$ and $in(v_i) = in(v_j)$
- **Strong conflict set (SC)**: $[v_i, v_j] \in SC$ if $out(v_i) = out(v_j) = in(v_i) = in(v_j)$ and, v_i and v_j are in a *critical path*.



	A1	A2	A3	A4	A5	A6	A7
IN	2	0	1	1	0	0	0
OUT	0	1	0	1	1	1	2

Conflicts:

- Late: $\{\overline{13}, \overline{24}, \overline{25}, \overline{26}, \overline{45}, \overline{56}\}$
- Early: $\{\overline{25}, \overline{26}, \overline{27}, \overline{56}, \overline{57}\}$
- Both: $\{\overline{25}, \overline{26}, \overline{56}\}$
- Strong: $\emptyset$

Figure 4.5: Conflict Sets

It is easy to see that, if an edge $[v_i, v_j]$ belong to any conflict edge set, then its orientation produces a circuit-free mixed graph G_M^k. The reason is that an edge is only in a conflict set if it has the same *inrank* or *outrank*, on both incident vertices, and a circuit is only possible if two vertices, with **different** *inrank* or *outrank* are connected.

The edge selection order is defined by the sequence of these sets. First we should select edges in *SC*, then *BC*, then *EC*, and finally in *LC*. *SC* has the highest conflict level because the orientation of an edge on this set will certainly increase $l(G_M^k)$. The next set, *BC*, contains edges with two types of conflict at the same time. Finally, we selecte edges from sets *EC* and *LC*. An example of the sets is depict at Figure 4.5.

A branch-and-bound approach is appropriate for the search of the optimal value of the enumeration of the set $P(G)$, using the lower and upper bounds described in Propositions 1 and 3. A branch-and-bound search node corresponds to a mixed graph G_M^k obtained from a transformation T^k. A search node can branch into up to two descendants, one for each orientation of the conflict edge selected to be transformed to an arc. Each branching descendant corresponds to a different mixed graph, constructed by the addition of the arc created by the orientation of the conflict edge. The two new mixed graphs will be $G_M(V, A^k \cup [v_i, v_j], E \backslash (v_i, v_j))$ and $G_M(V, A^k \cup [v_j, v_i], E \backslash (v_i, v_j))$. A^k describes the current set of arcs of the mixed graph G_M^k.

The branching process stops if the conflict edge sets are empty. When no more conflict edges exist, a feasible schedule for the problem has been found. In this case, the sched-

ule length will be $l(G_M^k)$. Each transmission of the aggregation convergecast tree can be scheduled at the time slot defined by $in(v_i)$ of the mixed graph G_M^k.

Assuming $l_{opt}(G_M)$ is the smallest schedule length found so far during the enumeration process, and because each search node represents a specific lower bound $l(G_M^k)$ that can be found in polynomial time (Proposition 2), we check, if the lower bound for the current node is greater or equal to the smallest found so far, and if yes, there is no reason to continue branching any further. This is because the addition of a new arc (from the transformation process) can only increase the lower bound. This criterion constitutes an effective method to prune the search space.

The exploration of the search tree can be performed in different ways. A simple one is to execute a Depth First Search (DFS) where, after a search node is branched, the search always continues in one of the branched search nodes, until a stop criterion is met. A second method uses a sort of guided search, where the branch-and-bound algorithm tries to guess what would be *better* branch to explore first, based on some heuristic. The complete process is described in Algorithm 3.

Complexity Analysis

The *Aggregation Convergecast Scheduling* algorithm is composed of initialization steps (Line 1 to Line 5) performed before the main search process (Line 6 to 24). The main step of the initialization is executed in Line 1, where the *TransformAggConvInstance* function creates the mixed graph G_M. This function is composed of three parts, each one with a particular run-time complexity. Initially it is necessary to create the mixed graph vertex set V_M using the aggregation Tree T_{AGG}. As the number of edges of the tree is equal to the number of vertices minus one, the run-time complexity of this first part is $\mathcal{O}(|V|)$. The second part consists of creating the arc set A_M from the precedences of tree T_{AGG}. The second part requires a run-time complexity of $\mathcal{O}(|V|^2)$ because for each vertex V we must check if there exists an arc with its neighbors. The worst case requires that we check against all other vertices. The third part produces the set E_M, which represents the interference conflicts between the transmissions. In order to check the interference, it is necessary to determine for each vertex in V, if its 1-hop neighbors have arcs coming from its 2-hop neighbors. This operation has a run-time complexity of $\mathcal{O}(|V|^3)$. Therefore, the run-time complexity of Line 1 is $\mathcal{O}(|V|^3)$. Lines 2 and 3 are simple operation executed in linear time $\mathcal{O}(|V|)$. Lines 4 and 5 are simple operations requiring $\mathcal{O}(|1|)$.

Algorithm 3: Aggregation Convergecast Scheduling

Input: Undirected Graph $G = (V, E)$ and Aggregation Tree T_{AGG}
Output: Schedule S

1 $G_M(V_M, A_M, E_M) \longleftarrow TransformAggConvInstance(G, T_{AGG})$
2 $UB \longleftarrow |V_M|$ /* upper bound for schedule */
3 $E_f \longleftarrow E_M$ /* set of edges */
4 $N_0 \longleftarrow CreateSearchNode(G_D(V_M, A_M), E_f)$
5 $Q \longleftarrow enqueue(N_0)$
6 **while** $Q \neq \emptyset$ **do**
7 $\quad N_k(G_D^k, E_f^k) \longleftarrow unqueue(Q)$
8 $\quad Ranks \longleftarrow CalculateInOutRanks(G_D^k)$
9 $\quad$ **if** $l(G_D^k) < UB$ **then**
10 $\quad\quad CE \longleftarrow CalculateConflictEdgeSets(Ranks)$
11 $\quad\quad$ **if** $CE = \emptyset$ **then**
12 $\quad\quad\quad$ **if** $l(G_D^k) < UB$ **then**
13 $\quad\quad\quad\quad UB \longleftarrow l(G_D^k)$
14 $\quad\quad\quad\quad S \longleftarrow GetInRank(Ranks)$
15 $\quad\quad\quad$ **end**
16 $\quad\quad$ **else**
17 $\quad\quad\quad \overline{ij} \longleftarrow SelectHighestConflictEdge(CE)$
18 $\quad\quad\quad N_{k+1} \longleftarrow CreateSearchNode(G_D^k(V_M, A_M^k \cup \overrightarrow{ij}), E_f^k \setminus \overline{ij})$
19 $\quad\quad\quad Q \longleftarrow enqueue(N_{k+1})$
20 $\quad\quad\quad N_{k+2} \longleftarrow CreateSearchNode(G_D^k(V_M, A_M^k \cup \overrightarrow{ji}), E_f^k \setminus \overline{ij})$
21 $\quad\quad\quad Q \longleftarrow enqueue(N_{k+2})$
22 $\quad\quad$ **end**
23 $\quad$ **end**
24 **end**
25 **return** S

From Line 6 to 24 we have the algorithm's core enumeration process. The run-time complexity depends on the maximum possible size of the search tree. As it is presented in Theorem 1, the cardinality of the set $\mathring{P}(G)$ is upper bounded by $\lambda(\mathring{P}(G)) \leq 2^{|E|}$, that is the maximum number of combinations of edge orientations. Therefore, the run-time complexity of this part is $\mathcal{O}(2^{|E_M|})$, a clear exponential run-time complexity. The conclusion is that *Aggregation Convergecast Scheduling* algorithm has a run-time complexity of $\mathcal{O}(2^{|E_M|})$.

4.5 Experiments and Discussion

A sequence of numerical experiments were carried out to explore both the aggregation tree selection and the results derived from the branch-and-bound solver for ACS. We generated

connected graphs generated for a fixed number of nodes n={40, 50, 60}, with varying number of edges by varying the transmission range, txr, and specifically for txr={0.20, 0.25, 0.30} distance units, with the nodes placed in a square area with side equal to one distance unit. A number of runs where produced for each parameter setting. The sink node is randomly selected out of the n nodes. Using these input graphs, aggregation trees are generated using α={1.00, 1.25, 1.50, 1.75, 2.00, 2.25, 2.50, 2.75, 3.00}. The α selected range encompasses our interest zone: the most probable values for which the combination of shortest path and minimum interference trees will produce short schedules. For the branch-and-bound scheduling, we used a timeout of 3,600s (1 hour) of computation time on a single processor of a Dual Core AMD Opteron(tm) 280 2.4GHz CPU. With the branch-and-bound timeout value used, and depending on the input graph, in some cases all runs in a group terminate, while in others no run can terminate within the given time. If a run could not terminate in the given time, we report the best solution found up to the termination of execution, noting that the optimal value could (given more processing time) have been lower.

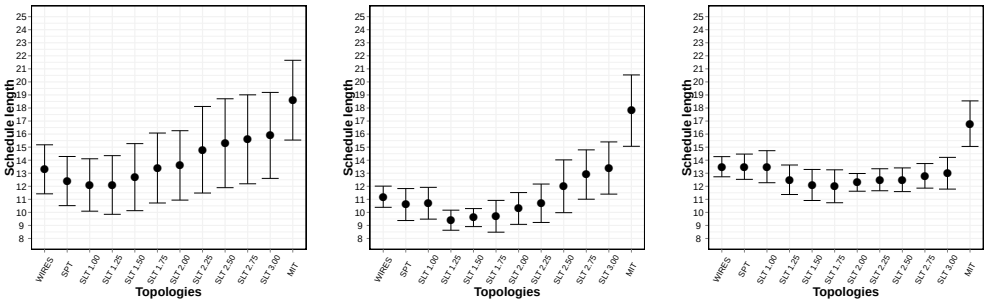
Figure 4.6 presents average and 95% confidence interval of the schedule produced by trees generated using several α values. The points for *WIRES* correspond to the results of the WIRES algorithm [76] given the same input, used here as a baseline of current state-of-art solution for the ACS problem. The points for *MIT* correspond to the results of the use of the Minimum Interference Tree (MIT), also used here for comparison purposes.

Figures 4.7 and 4.8 present the results of *Tree Size* and *Interference Weight Count*, as the number of nodes, transmission range and α change. The average results of SPT and MIT are also used for purposes of comparison.

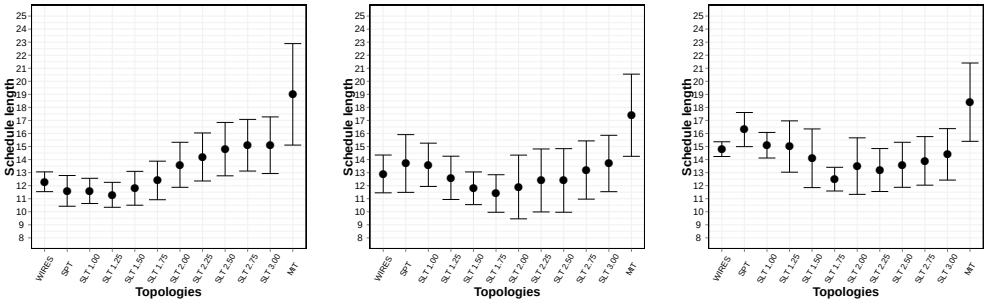
Figure 4.9 presents the change of the schedule length as α changes with respect to the schedule lower bound. The points represent the average schedule length for a given α.

4.5.1 Balancing Precedence and Resource Constraints

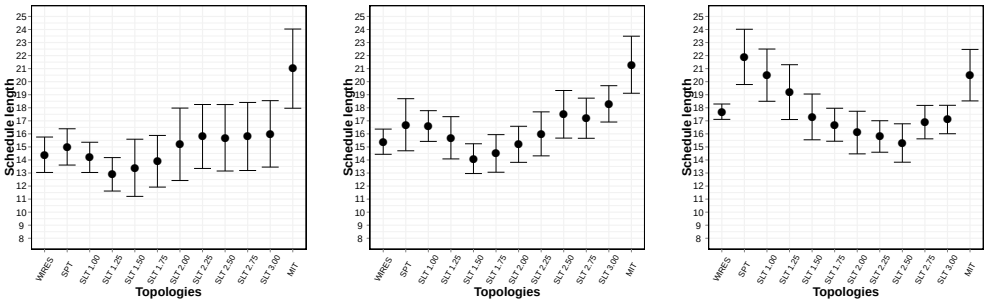
We explained in the beginning that the aggregation convergecast scheduling problem is composed of two types of constraints, precedence and resource/interference constraints. Therefore, we must balance both requirements to obtain results closer to optimal, based on the control afforded to us by the α parameter. Figure 4.6 shows that there indeed exists an α such that the combination of both sets of constraints produces a schedule length smaller than



(a) $n = 40, txr = 0.20$ (b) $n = 40, txr = 0.25$ (c) $n = 40, txr = 0.30$



(d) $n = 50, txr = 0.20$ (e) $n = 50, txr = 0.25$ (f) $n = 50, txr = 0.30$

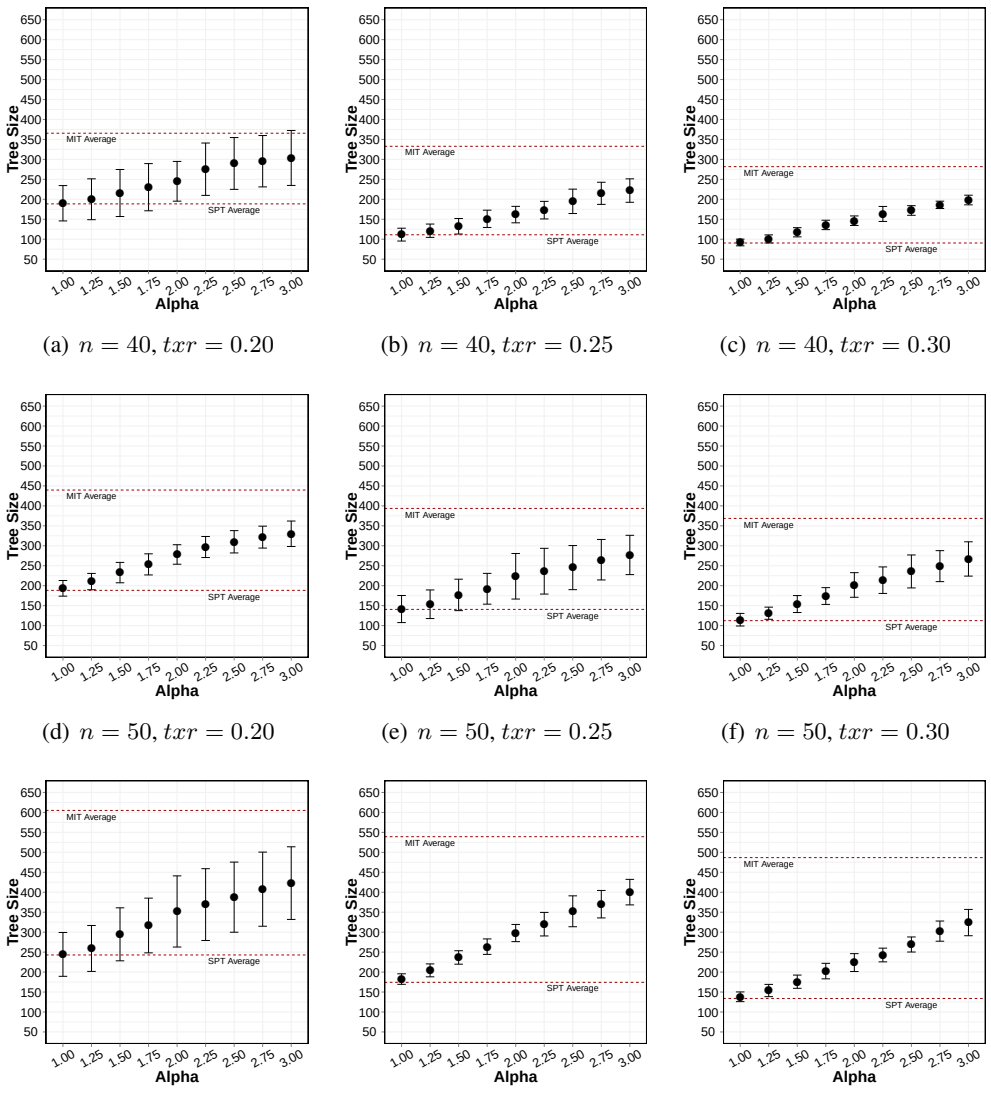


(g) $n = 60, txr = 0.20$ (h) $n = 60, txr = 0.25$ (i) $n = 60, txr = 0.30$

Figure 4.6: Schedule generated by each tree created using different α

the schedule length obtained when only one restriction (precedence or resource constraint) is minimized. The results also show that there exists a valley around α's optimal value.

This optimal α value depends on the input instance: the size of the network in terms of vertices and edges (the more the transmission range the more the edges). The parameter that seems to cause greater shift on the optimal value of α is the transmission range (hence, edge density) because it captures the impact of interference, even though higher density can decrease the impact of the precedence component; while if the transmission range is smaller, it will decrease the interference component, and increase the precedence component (i.e., longer paths), but not at the same rate. It is noticeable in Figures 4.6(a) to 4.6(i) that the



(a) $n = 40, txr = 0.20$ (b) $n = 40, txr = 0.25$ (c) $n = 40, txr = 0.30$

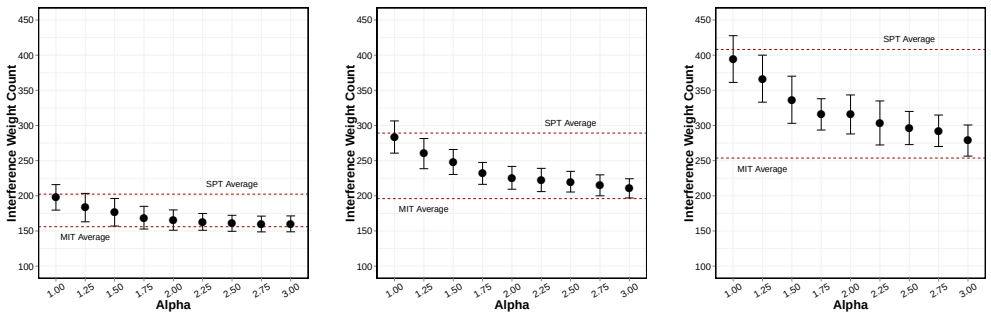
(d) $n = 50, txr = 0.20$ (e) $n = 50, txr = 0.25$ (f) $n = 50, txr = 0.30$

(g) $n = 60, txr = 0.20$ (h) $n = 60, txr = 0.25$ (i) $n = 60, txr = 0.30$

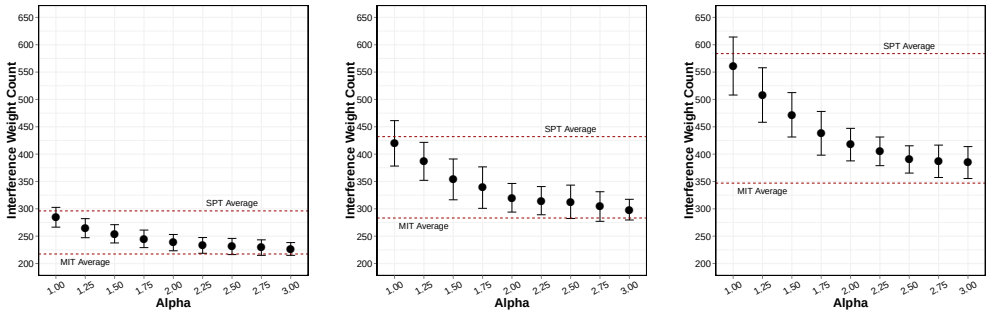
Figure 4.7: Tree Size of trees using different α

best α shifts right (increasing in value) following the increased importance of the resource constraint in dense graphs. On the other hand, in sparse graphs, smaller α produces better schedule lengths. Therefore, for sparse graphs, SPT is a competitive solution, because the weight of the precedence constraints is higher.

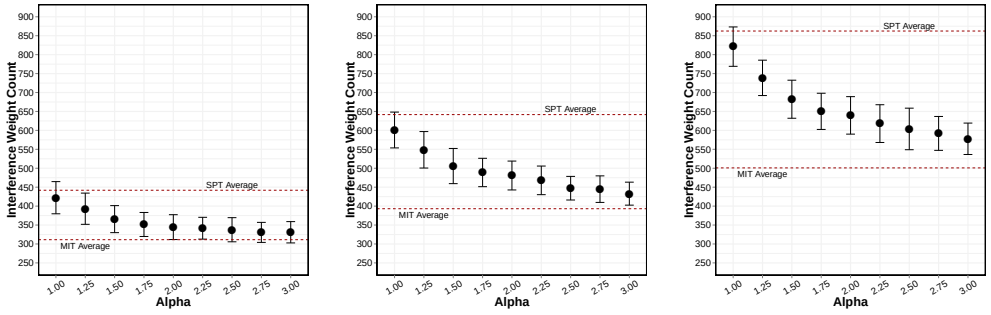
The effect of the different values for α on the tree size are shown in Figure 4.7 where the two dotted lines indicate the average tree size of SPT and MIT. These are the limits expected for the input parameters. The range between the maximum and minimum values do not change a lot as the transmission range (and hence, density) increases, however it is clear that the tree size will be smaller when there is a longer transmission range. Furthermore,



(a) $n = 40, txr = 0.20$ (b) $n = 40, txr = 0.25$ (c) $n = 40, txr = 0.30$



(d) $n = 50, txr = 0.20$ (e) $n = 50, txr = 0.25$ (f) $n = 50, txr = 0.30$

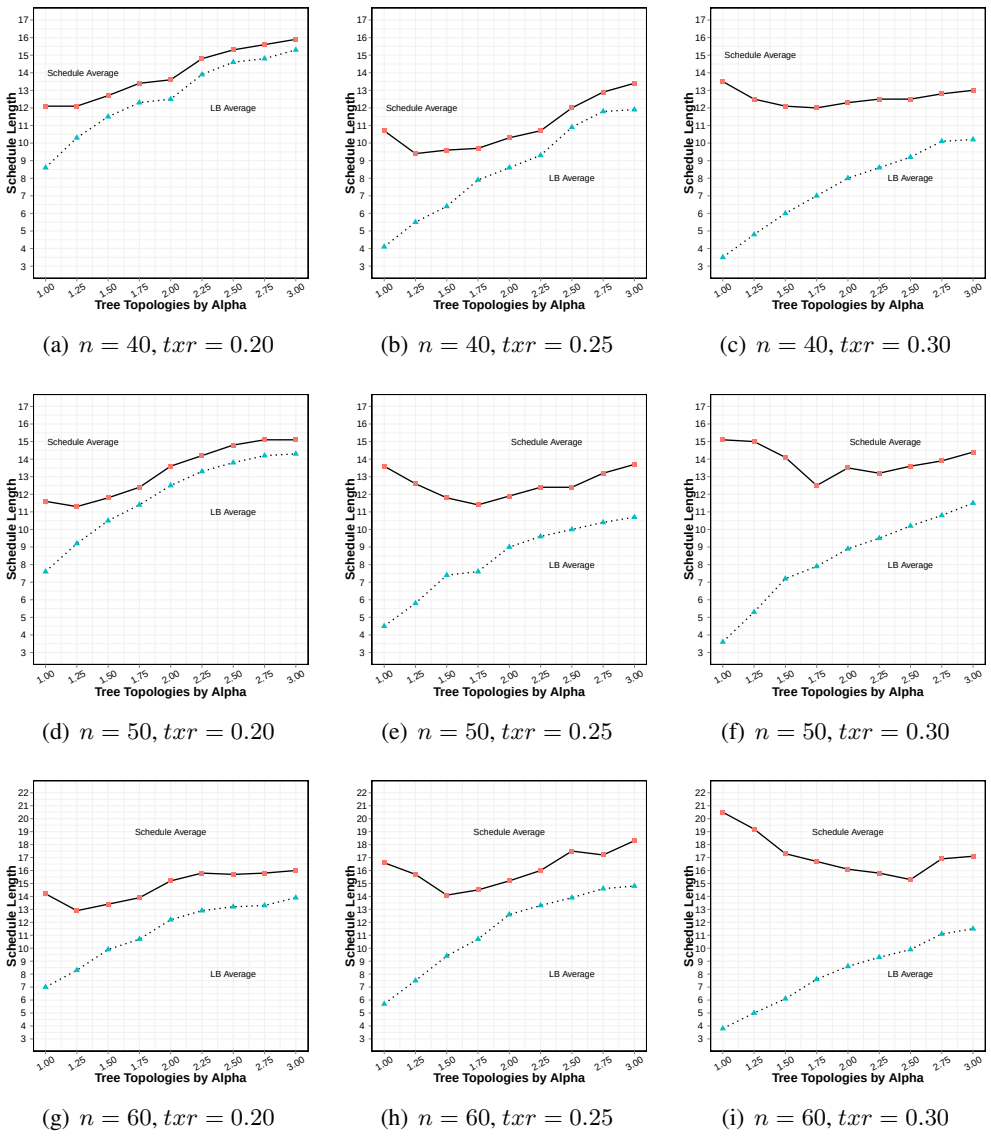


(g) $n = 60, txr = 0.20$ (h) $n = 60, txr = 0.25$ (i) $n = 60, txr = 0.30$

Figure 4.8: Interference Weight Count for Trees using different α

if the transmission range is kept constant as the number of nodes vary, the SPT average changes little, while MIT average shifts significantly. Given the same number of nodes and transmission range, the modification of the α parameter has an almost linear impact on the tree size.

Figure 4.8 illustrates the influence of α selection on the interference weight count with respect to two aspects. First, the gap between interference weight averages for MIT and SPT topologies widens as txr increases. This effect is far less pronounced on the tree size. The second one is (differently from the almost linear change on the tree size) a non-linear impact on the interference weight when α changes. This last aspect means that a small



(a) $n = 40, txr = 0.20$ (b) $n = 40, txr = 0.25$ (c) $n = 40, txr = 0.30$

(d) $n = 50, txr = 0.20$ (e) $n = 50, txr = 0.25$ (f) $n = 50, txr = 0.30$

(g) $n = 60, txr = 0.20$ (h) $n = 60, txr = 0.25$ (i) $n = 60, txr = 0.30$

Figure 4.9: Comparison of average schedule obtained and lower bound for different α

variation of the α parameter is enough for a large variation on the amount of transmissions that could be blocked (to satisfy interference constraints).

In the set of graphs presented in Figure 4.9, the lower line represents the contribution of precedence constraints on the schedule length, understood as the baseline of what would have been the schedule length if no interference (resource) constraints existed[2]. The upper line is the actual schedule obtained through our runs. The gap between the lines represents the contribution of the resource (interference) constraints on schedule length. Clearly, de-

[2]An alternative view of the same is that this line would represent the schedule length achieved if multiple frequency channel transceivers were used by each node.

pending on the value of α selected, SPT and MIT will be combined such that this gap is somehow *compressed*, otherwise the upper line would just follow the lower line by a constant distance, from the smallest to the highest α used. When the influence of resource constraints is smaller, as it happens with small txr, the gap is small and constant. The schedule average line follows closely the trend of the lower bound.

An important open question is what should be the *"best"* value for α such that the schedule is minimized, given a number of nodes, the location of the sink, their transmission ranges and the area of deployment. A closed form for the optimal α has proved to be beyond our abilities for the time being. However, two aspects may be used to confine the search for the best α. The first is that the best α value is higher for higher interference weight. The second aspect possible to explore is the presence of a single *"inflexion point"* on the schedule length around this best value. As soon as the trend of the schedule length decreasing as α increases, we can be fairly certain that we have outside the range in which the optimal α is located. These two aspects could be the basis of an "outer loop" for an iterative optimization strategy the best α value can be found, and consequently, the optimal schedule length. Finally, we remark a significant caveat: the timeout used for the branch-and-bound and the fact that most instances did not fully terminate within the alloted time means that the absolute results, especially for dense graphs, could change if more processing resources were allocated. This could cause the best α to shift in some cases. Nevertheless, the results could only improve, decreasing even more the schedule length.

4.5.2 Generalization of Aggregation Convergecast Scheduling Model

In Section 4.4, we explain how to model aggregation convergecast scheduling problem as a Mixed Graph Coloring problem, and how precedence and resource constraints fit on this model. In this part, we explain the equivalence of aggregation convergecast scheduling to similar (with the same constraints) scheduling problems. The literature of aggregation convergecast research seems dissociated of scheduling theory. An association is not found in the literature and it was never addressed by researchers.

Scheduling Theory covers a vast literature [83]. Let's focus initially in one well known problem: the Deterministic Job-Shop Scheduling. A Job-Shop consists of a multi-stage processing system where a set of machines $M = \{M_1, M_2, \cdots, M_m\}$ has to process a set of given jobs $J = \{J_1, J_2, \cdots, J_n\}$. Each job J_i is composed of n_i ordered operations $(O_{(i,1)}, O_{(i,2)}, \cdots, O_{(i,n_i)})$. Associated to each operation O_{ij} is a processing requirement

p_{ij}. A release time r_i may be associated to the first operation of job J_i, indicating when this job can start. Each operation O_{ij} is associated to a set of machines $\mu_{ij} \subseteq \{M_1, \cdots, M_m\}$. O_{ij} can be executed on machines in μ_{ij} set. For the specific case of Job-Shop Scheduling, all μ_{ij} are one element set, indicating a dedicated machine to execute a specific operation O_{ij}. A technological route is a sequence of machines $(M_{i,1}, M_{i,2}, \cdots, M_{i,n_i})$ that is used to process the job J_i. Some conditions are assumed:

- At any time, each machine $M_j \in M$ either processes one and only one job from the set J or is idle.
- At any time, each job $J_i \in J$ is either been processed by one machine from set M, or is waiting to be processed, or is already completely processed.
- The technological route for processing each job $J_i \in J$ is fixed before scheduling.

The objective of the scheduling is to determine a sequence for the set of all operations $L = \{L_1, \cdots, L_q\}$ on the corresponding machines for which the value of the given objective function F is minimal. A usual objective is to minimize the makespan. The example of Figure 4.10 presents a job shop of 3 jobs (J_1, J_2, J_3) and 4 machines (M_1, M_2, M_3, M_4). The job J_1 is composed of operations $\{O_{(1,1)}, O_{(2,1)}, O_{(3,1)}\}$, J_2 has operations $\{O_{(2,2)}, O_{(1,2)}, O_{(4,2)}, O_{(3,2)}\}$, and J_3 has $\{O_{(1,3)}, O_{(2,3)}, O_{(4,3)}\}$ operations.

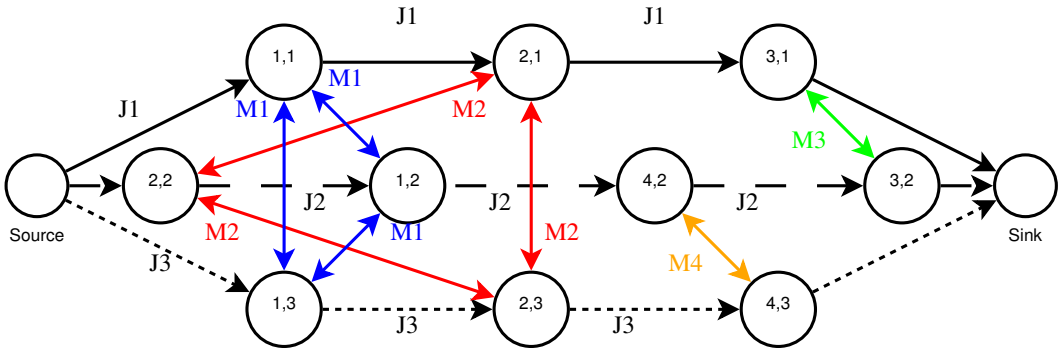


Figure 4.10: Disjunctive Graph for Job-Shop problem

It is not hard to see that the job shop scheduling problem have also precedence and resource constraints. The precedence constraints is defined by sequence of operations $(O_{(i,1)}, O_{(i,2)}, \cdots, O_{(i,n_i)})$ that each job J_i has to follow. The resource constraints is established by the impossibility that a given machine M_j execute more than one operation O_{ij} per time. This machine/resource restriction is expressed by a Disjunctive Graph using a clique among all the operations that must be executed by the same machine. In Figure 4.10, the set of jobs

$\{(2,1),(2,2),(2,3)\}$ are all executed by machine M_2. The Disjunctive Graph expresses the same restrictions conveyed by the extended conflict graph (Figure 4.4) of the aggregation convergecast scheduling problem.

Both, aggregation convergecast scheduling and job shop scheduling induce mixed graphs which represent restrictions of each problem. In principle, both problems can be scheduled by an algorithm similar to the branch-and-bound algorithm presented before. However, each problem induces different mixed graph classes. For instance, each machine of the job shop scheduling create disjointed cliques, representing the resource constraint of the machine. Besides, each machine has no relation with another machine. In aggregation convergecast, the resource constraint created by use of wireless media is blurred. An operation in Job-Shop scheduling is associated to a single machine (μ_{ij} set is unitary), which may not be true for aggregation convergecast. Another different is that Job-Shop Scheduling has disjointed technological routes. They are independent paths of precedence constraints that only converge in the last step. By his part, Aggregation Convergecast precedence paths forms a tree, merging different branches along the way, instead of only in the end. Therefore, even though some similarities exist, both problems induce different mixed graphs.

The set μ_{ij} of operation O_{ij} is not necessarily unitary, but can be a subset of the machines available. If more than one machine is required at a time by operation O_{ij}, the problem is called **Multiprocessor Task Scheduling** (MTS). A clarification is necessary. There are two categories of MTS problems [55]: the first considers that a job needs to be executed by a given number of processor simultaneously, but *choice of the processor is open*; the second category imposes that every job *requires a number of dedicated processors* [3]. Aggregation Convergecast falls on the second category.

In [15], Brucker presents a classification for classes of scheduling problems. A problem can be categorized according to a three-field classification $\alpha \mid \beta \mid \gamma$, where α specifies the machine environment, β expresses the jobs characteristics, and γ determines the optimality criterion. According to this classification, the aggregation convergecast is a $MPT_m|intree, p_i = 1|C_{max}$ scheduling problem. The parameters of this classification is explained in Tables 4.1 and 4.2. Studies of this problem can be found in [3, 23, 36, 70].

An aggregation convergecast scheduling problem can easily be transformed into a machine-job representation. The machine in the Multiprocessor Task Scheduling represents the con-

[3] The first problem is known as $P_m|set_j|C_{max}$, while the second is $P_m|fix_j|C_{max}$, [23]

Table 4.1: Scheduling Class Parameters

Symbol	Description
MTP_m	Multiprocessor tasks environment with m machines
$intree$	The precedence relations between jobs forms an inward rooted tree
$p_i = 1$	The processing time of a task is 1 (*Unit-Time Processing*)
C_{max}	The scheduling objective minimizes the makespan $max\{C_i \| i = 1, \cdots, n\}$, where C_i represents the finishing time of job J_i

Table 4.2: Scheduling Implicit Parameters

Symbol	Description
$\beta_1 = \emptyset$	No preemption is allowed
$\beta_5 = \emptyset$	No deadline is specified for each job J_i
$r_i = 0$	The release time of all jobs is assumed to be at the beginning of the schedule
$n_i = 1$	Each job J_i is composed only one operation $\{O_{i,1}\}$

flict for the execution of each task. This conflict is represented by the edges and arcs on the extended conflict graph. An immediate equivalence is to transform each edge and arc into an independent machine, as described in Figure 4.11. The number of machine can be reduced by decomposing the graph into cliques, and naming each clique as a machine, as done in Figure 4.12, and described in Table 4.3.

4.6 Previous Work

Several researchers have addressed the ACS problem [24, 38, 73, 76, 114]. The decomposition we adopt of ACS into an aggregation tree construction (using some heuristic) followed by a feasible schedule construction (using another heuristic) is the sequence adopted by a number of papers [24, 76, 112]. Typically, the tree construction is a form of expressing precedence constraints with limited (if any) representation of the interference constraints, while the scheduling stage expresses the impact of interference constraints. An explicit simultaneous use of *both* precedence and resource/interference constraints has not yet been explored. The use of an SPT as the aggregation tree reflects the concern of minimizing the effects of the precedence constraints on the schedule length, but it is oblivious to the resource constraints.

We compared our approach with the recent state-of-the-art works in the literature, [76], where the authors present a heuristic based on using BSPT (Balanced Shortest Path Tree)

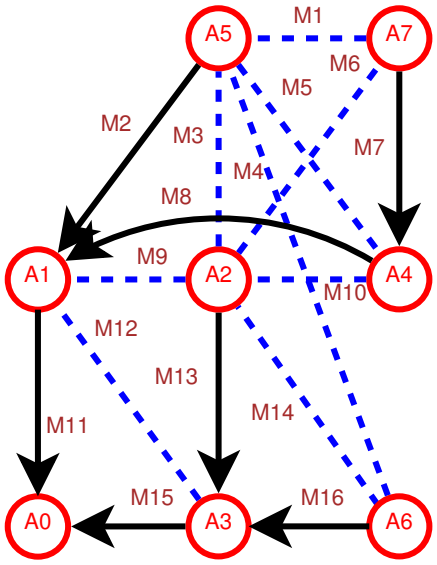


Figure 4.11: Simple job-machine conversion

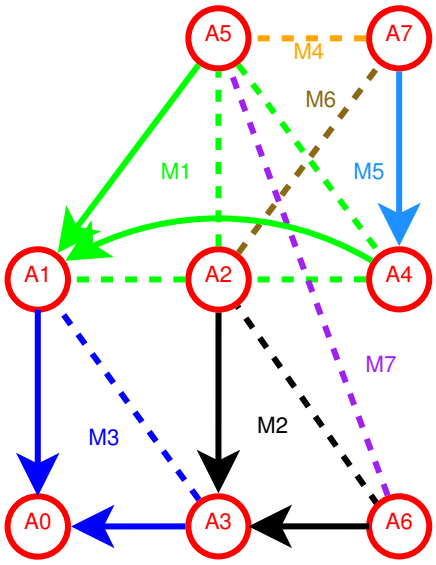


Figure 4.12: Job-machine conversion by clique decomposition

Task	**Machines**	**Prec**
$A0$	$M3$	$\emptyset$
$A1$	$M1, M3$	$A0$
$A2$	$M1, M2, M6$	$A3$
$A3$	$M2, M3$	$A0$
$A4$	$M1, M5$	$A1$
$A5$	$M1, M4, M7$	$A3$
$A6$	$M2, M7$	$A3$
$A7$	$M4, M5, M6$	$A4$

Table 4.3: Multiprocessor Task Equivalence

as the aggregation tree. The idea is to select a SPT such that it balances the number of children among the parents at each graph layer, whereby the number of children is a proxy of the potential interference. While this approach has the potential to decrease the overall interference weight, its restriction to particular SPT varieties precludes additional improvement, given the SPT pathologies outlined in Chapter 3.

The combination of precedence and resource constraints with the objective of minimizing the makespan through a merging of SPT and MIT trees (and therefore using more than one criterion to optimize the aggregation tree) has been recognized by other authors as well. In [51], a bi-criteria optimization [86] is used to combine SPT and MIT. The authors formulate the optimal aggregation tree construction as a bi-criteria optimization where, given a threshold on the maximum node degree, the objective is to minimize the maximum number

of hops (radius) of the tree. This problem is known as the Degree-Bounded Minimum Diameter Spanning Tree [68]. Initially, a backbone tree is constructed by arbitrarily choosing a local root from hexagon grid cells. These local roots are connected using a BFS (Breast First Search) approach, starting from the sink node. This constitutes the SPT side of the approach. After that, local spanning trees are created in each cell with the non-root nodes such that in each cell no node exceeds a maximum degree Δ^*. The spanning trees are constructed using a edge weight similar to $X_{edge}(e)$ in Figure 4.2(a) from Section 4.3. Even though our work and [51] have similarities, the objectives are different. The objective of [51] is to create a tree for a pipelined schedule, which removes the concern about precedence constraints. Therefore, the radius of the resulting tree is not a decisive concern to their objective. For comparison purposes, we run some simulations, using the scheduler described in [52] on a set of input graphs.

Table 4.4: SLT vs. BDMRST (200 nodes, $\alpha = 1.25$, single channel)

BDMRST Schedule	SLT Schedule	BDMSRT Radius	SLT Radius	BDMRST Max Degree	SLT Max Degree
14	16	28	13	6	13
13	16	16	17	6	9
16	18	18	17	8	13
15	17	26	17	6	8
12	14	23	14	5	9

The results on Table 4.4 show that for an $\alpha = 1.25$ the tree created using BDMRST produces smaller schedules length. However, this results is for a pipelined schedule, where precedence constraints do not exist. If precedence is a requirement, as in ACS, the schedule length would be very different. The column with the BDMRST radius reveals a deep tree. If this deep tree was used for ACS, the BDMRST schedule would have been at least as long as the reported tree radius. Interestingly, it is possible to obtain a SLT *similar* to BDMRST by determining a suitable value for α. For example, results for an $\alpha = 2.25$ are presented in Table 4.5.

Table 4.5: SLT vs. BDMRST (200 nodes, $\alpha = 2.25$, single channel)

BDMRST Schedule	SLT Schedule	BDMSRT Radius	SLT Radius	BDMRST Max Degree	SLT Max Degree
13	11	24	26	7	8
15	16	21	28	7	14
12	11	26	22	6	6
13	13	27	25	5	8
13	11	23	24	6	7

In [76], Malhotra *et al.*, alongside the introduction of WIRES against which we compared the numerical results in the previous section, they also remark that for a given routing tree,

the lower bound on the schedule length is $max_{i \in V}(\xi_i + h_i)$, where ξ_i and h_i are the number of children and hop distance from the sink, respectively, for node i. A heuristic is used to schedule transmissions from the nodes toward the sink. The idea is to rank all eligible nodes in decreasing order of their weights (number of non-leaf neighbors). A higher weight gives a higher relative priority to a node to be scheduled in the current time slot over other eligible nodes. Such heuristic has the effect of releasing (transmitting) earlier the node that is blocking more nodes from transmitting. It also creates a collision-free schedule.

In [52], Ghosh *et al.*address the multi-channel scheduling problem. Given a tree T from a graph G, and K orthogonal frequencies, a frequency is assigned to each one of the receivers, and a time slot to each of the edges in T, such that the schedule length is minimized. The part of [52] relevant to our work for comparison purposes is the time slot assignment. The authors opt for a greedy time slot assignment scheme for the whole network. The deployment region is divided into a set of grid cells. Each cell needs γ_i time slots. If the set of time slot γ_i represents a unique color, then the whole network can be schedule using at most four different colors. As precedence is not required for the schedule, the total number of time slots required is 4 times the maximum number of slots in any set. Other works on aggregation convergecast pose different requirements than what we assume here. For example, one group [37, 93] aims to perform aggregation convergecast scheduling, but with variable transmission power, such that the original communication graph may change. Another group [72, 112] seeks a distributed algorithm to execute the aggregation and scheduling.

Finally, we point out the uniqueness in our approach of obtaining the MIT on the way to producing the SLT. By adopting a "directed" link activation model (instead of a node activation model) to capture the interference metric, we have formulated the scheduling problem in a way specifically suitable to ACS. Our MIT topology differs from works such as [13, 45, 108] where an initial minimum interference topology was attained by controlling transmission power. In ACS, the MIT is an *overlay/logical topology* whose selection reduces the interference for the purposes of the schedule length. Conceivably, the MIT may be constructed after a minimum interference topology (using transmission power control) has been first selected. However, our MIT algorithm is suitable for ACS compared to flow-based minimum interference routing (where individual flows are preserved and not aggregated) [44, 67], because, the aggregation implies non-conservative flow model, which is clearly not the case with conservative flow-based minimum interference algorithms.

4.7 Conclusions

The precedence and resource constraints were explored in depth. Existing solutions prioritize one constraint over another instead of approaching the problem as a case of bi-criteria optimization. We propose a method to combine both constraints such that the resulting logical topology is a synthesis of properties of a shortest path tree (minimum precedence constraint tree) and properties of a minimum interference tree (minimum resource constraint tree).

Additionally, it is shown that aggregation convergecast scheduling can be modeled as a mixed graph coloring. This led us to the definition of an extended conflict graph representation for the aggregation convergecast. Arcs represent the aggregation paths, and edges represent interference conflict between two transmissions. The chromatic number and relevant properties for the mixed graph are presented. A branch-and-bound method to obtain the schedule length is developed. Extensive simulation results show that the right balance between precedence constraints and resource constraints produces schedule lengths shorter than current state-of-the-art heuristics can attain. We also observe that the aggregation convergecast scheduling is focused to a single collection of all sensor data to the sink, and hence centered on completing the collection in the shortest amount of time.

Chapter 5

Pipelined Aggregation Convergecast

5.1 Introduction

The *aggregation convergecast scheduling* problem is concerned with determining the best aggregation tree in terms of schedule length required to collect a single snapshot of data from the nodes to the sink. Essentially, the output of aggregation convergecast scheduling is a schedule of transmissions and a corresponding aggregation tree.

In aggregation convergecast, leaf nodes transmit their measurements to their parent nodes. Interior nodes of the tree perform the aggregation operation on the values arriving from their children (and their own value) and transmit the aggregation result to their own parent node. The restriction of forcing parent nodes to wait for the results from *all* their designated children to be received before they produce and transmit the aggregation result, is called *precedence constraint*.

The solution approach used in most of the relevant literature is to decompose the problem into two phases: the first one constructs an aggregation tree (i.e., determines the routing), and the second determines the transmission time of each node (i.e., scheduling). Both phases rely on heuristics. Two points are (sometimes implicitly) assumed: (a) that it is necessary to define an aggregation tree *first* in order to obtain a schedule (the two phase approach), and (b) the (strict) enforcement of *precedence constraints* as having to be satisfied within one schedule cycle.

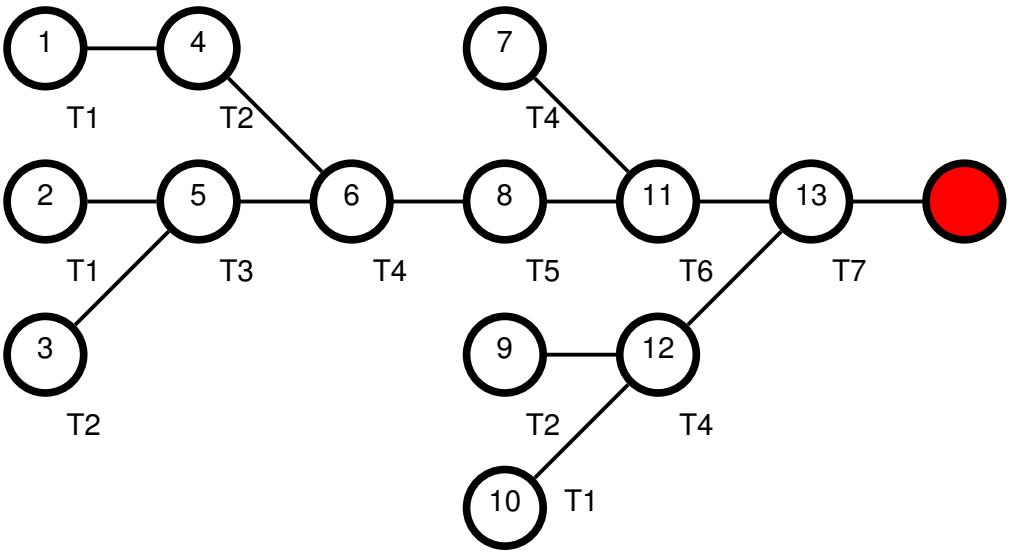
Aggregation convergecast schedule has, thus far, been interested in a single (isolated) collection of all sensor data to the sink, and hence focused on completing the collection in the shortest amount of time. While this approach is desirable for some classes of applica-

tions, it does not serve other applications, such as the continuously running data aggregation queries. Continuously running queries may prefer the ability to collect frequent samples from the entire sensor field, to allow a finer temporal reconstruction of the observed phenomenon. Such applications demand higher sampling rate, i.e., a higher rate of snapshots being collected per unit of time, and hence higher throughput data collection. In fact, we may be willing to accept a longer lead–in time for the first snapshot to be collected as long as the collections can be performed at a higher rate. To this end, we consider relaxing assumption (b) mentioned in the previous paragraph.

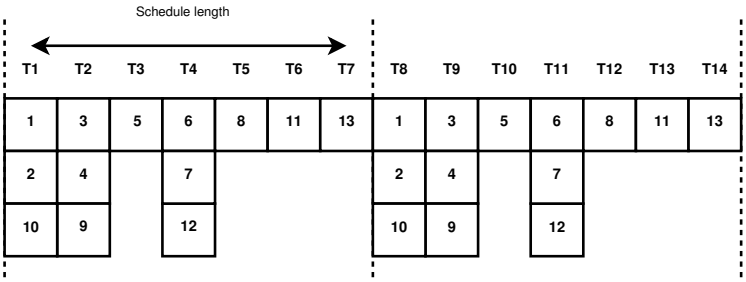
Specifically, the solution proposed in this Chapter considers satisfying the totality of precedence constraints over a timespan of multiple, consecutive, scheduling cycles. In contrast, the characteristic of a single data collection cycle is that within each scheduling cycle a single snapshot is allowed to be transmitted. Compared to previous schemes, our main idea is to produce a short scheduling cycle. However, a single such cycle will be insufficient to complete the data aggregation of a single snapshot of data from all sensors. Multiple such cycles are needed to complete a single snapshot aggregation, but the upside is that within a scheduling cycle, several data aggregation snapshots may be collected/aggregated in parallel, i.e., a form of pipelining of many snapshots by interleaving their collection over time.

The intuition is that pipelining should be possible because of the extent that spatial reuse of the medium in a multi-hop wireless network is possible. This ”spatial” dimension can be roughly thought as representing stages of a pipeline. An example of the benefits of using pipelining is presented in Figure 5.1. The throughput of $1/7$ (Figure 5.1(b)) without using pipelining increases to $1/4$ (Figure 5.1(c)) using pipelining. For example, in Figure 5.1(c), node 13 cannot transmit the aggregation (for one particular snapshot) within a scheduling cycle, as node 12 (which feeds 13) is set for transmit (in schedule order) after node 13, hence the precedence constraint of 11 and 12 transmitting to 13 before 13 can transmit is satisfied across two schedule cycles. The tree topology depicted in Figure 5.1 is, relatively speaking, trivial to pipeline. As we will see in subsequent sections, our pipelining technique applies to arbitrary connected graphs.

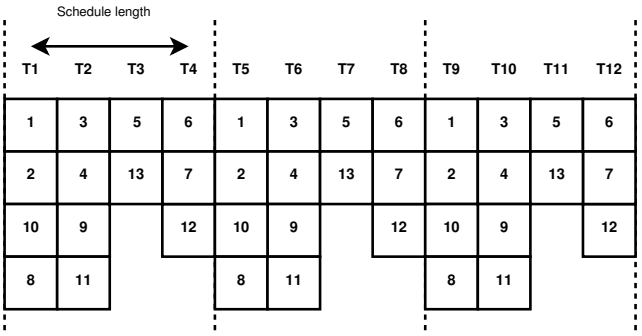
Our proposal involves the unconventional approach of constructing the schedule before finalizing the exact form of the precedence constraints, i.e., before determining the data aggregation tree, which in turn requires that the schedule construction phase guarantees that every node can reach the sink. We compare our results using pipelining against a previously



(a) Communication Graph



(b) Non-Pipelined Schedule (1/7 snapshots/slot)



(c) Pipelined Schedule (1/4 snapshots/slot)

Figure 5.1: Throughput improvement using pipelining

proposed algorithm that also uses pipelining, as well as against an algorithm that, although lacking pipelining, exhibits the ability to produce very short schedules. The results confirm the potential to achieve a substantial throughput increase at the cost of increased latency.

The remaining of the Chapter is organized as follows. In Section 5.2, related work is presented. Section 5.3 contains the aggregation convergecast related definitions. The proposed

algorithm is described and discussed in Section 5.4. Extensive simulations and consequent results are presented in Section 5.5. Energy consumption is addressed in a separated part. Finally, Section 5.6 concludes the Chapter.

5.2 Related Work

We consider three relevant aspects for the purposes of study in the current Chapter: the relaxation of precedence constraints in a single cycle, the sequence (order of steps) to obtain a solution (node scheduling first, aggregation tree later); and, the use of pipelining. With respect to the first aspect, most works explicitly force parent nodes to only transmit after they have received the transmission from all their designated children in the same cycle [76], while other works use the same restriction implicitly [4, 111]. On the second aspect, to the best of our knowledge, all previous algorithms decide first about aggregation tree, and later about the node's transmission time/schedule. On the third aspect, we did not locate in the literature a solution proposing pipelining, except for [52]. Even though, Ghosh *et al.* use pipelining, they first commit to an aggregation tree, and create a pipelined schedule over this tree. We did not find a single work that jointly combines the three aspects mentioned.

Since the closest to our work is [52], we provide a summary of its operation. The aggregation tree constructed is called a Bounded-Degree Minimum-Radius Spanning Tree (BDMRST). The aggregation tree uses bi-criteria optimization to combine a Shortest Path Tree and a Minimum Interference Tree. The optimal aggregation tree construction is formulated as follows: given a threshold on the maximum node degree, the objective is to minimize the maximum number of hops (radius) in the tree. Subsequently, the aggregation tree is used as the basis for multi-channel scheduling. The WSN deployment area is divided into a set of square grid cells. First, frequencies are assigned to receivers of the tree on each cell, then a greedy time slot assignment scheme is employed for each cell. The scheduling algorithm does not require the precedence constraint, instead, a pipeline is established for the sink to receive aggregated data from all nodes.

5.3 Preliminary Definitions

Definition 5.3.1. *Let snapshot s^t be defined as the union of the sensed values produced by n sensors at a particular time instant t.*

Definition 5.3.2. *The collection delay Δ_c is the difference between the time t when a snapshot (sensing by all nodes simultaneously and across all nodes of the network) is taken and the time when the sink has received all the data related to this snapshot.*

Definition 5.3.3. *Precedence constraint is the restriction that once a node transmits, for the purpose of a particular single snapshot data collection, it can no longer be the destination for any transmissions related to this same snapshot.*

The way precedence constraints were captured in previous works resulted in significant reduction on the solution space of possible schedules, because of only allowing solutions where the collection latency, Δ_c, (from snapshot "capture" to arrival at the sink) was the same as the length of the schedule cycle, l. For pipelined aggregation convergecast we introduce some additional definitions:

Definition 5.3.4. *An aggregation convergecast employs* pipelining *if the sensors can begin to transmit data of the next snapshot $s^{t_{k+1}}$ before the previous snapshot s^{t_k} has been completely received by the sink.*

Definition 5.3.5. *Inter-snapshot delay δ_s is the time difference as perceived by the sink between a snapshot s^{t_k} and the next snapshot $s^{t_{k+1}}$ being completely received.*

The rate at which snapshots are created should match the rate at which they are delivered to the sink. Hence, δ_s is a property of the particular schedule which, in turn, defines how frequently the snapshots can be generated. Also, after an initial transient time where no snapshot has completely been received by the sink, we reach a steady-state, whereby the pipeline is kept utilized and reception of snapshots is periodically completed at the sink. It is trivial to note (by contradiction) that in steady-state the inter-snapshot delay is equal to the schedule length $\delta_s = l$.

The purpose of pipeline is to have more than one snapshot propagating through the network during each schedule period, and hence $\Delta_c \geq \delta_s (= l)$, noting that in previous works these three quantities were equal.

Definition 5.3.6. *Aggregated Throughput C is the ratio of the amount of data of one snapshot over the inter-snapshot delay, $C = \frac{n}{\delta_s}$. It can also be understood as the rate of completed data snapshot collections per unit of time, as perceived by the sink node.*

The solution strategy we advocate for pipelined aggregation convergecast, tries to reduce l by constructing a "tight" schedule and subsequently constructing an aggregation tree that

"fits" with the schedule. This inversion of order (compared to previous work on aggregation convergecast) comes with particular requirements for what interference model can be used during the schedule construction phase. Two such models are commonly used: node activation and link activation models [54]. It is commonly understood that in the node activation model, when a node is scheduled to transmit, there should be no transmission by **any** of its 1st and 2nd hop neighbors. The appeal of the node activation model is that we do not need to commit who is the intended recipient (1st hop neighbor) of the transmission node, i.e., it is a model that allows us to decide on a schedule before we decide on routing. This flexibility comes at the cost of low throughput [54].

In the link activation method, an edge (without direction) or an arc (with direction) is what is scheduled. In the case of an edge, no direction is designated. Then, it is necessary to block edges up to 2 hops away from both vertices from being active at same time to avoid collisions. This form of blocking is even higher then in the node activation model. More useful is the case of directed (arc) link activation. In this case, the arc's source node transmits, and must be received in a collision–/interference–free manner **only** by the arc's destination node. Hence, arc link activation models achieve higher throughput than node activation [54]. It comes also at a price. Namely, it is necessary to commit first to a desired direction (routing) before deciding when an arc is active (scheduling).

In the proposed approach, because routing follows the schedule construction, we will not use the link activation model and will opt for a node activation model with full knowledge of the fact that doing so risks the reduction of overall throughput. As we will see in the following, any such reduction is apparently compensated by the throughput increase gained by pipelining.

In summary, we define the schedule $S = \{S(1), S(2), \cdots, S(r), \cdots, S(l)\}$ be the sequence of concurrent transmissions $S(i)$ taking place in slot i. $S(i)$ is the set of nodes that transmit in slot i, also called the set of *active* nodes in the i–th slot. Assume $v_j \in S(i)$ is a node transmitting in slot i, we denote by a_{v_j} the outgoing arc towards the sink of the transmission of node v_j. It follows that the aggregation tree is defined by $T = \bigcup_{v \in V - \{s\}} a_v$. In previous aggregation convergecast schemes, a_v was fixed by means of a pre-computed aggregation tree, while in the pipelining proposed here, a_v is determined by a particular spanning tree construction phase subsequent to the scheduling phase.

An additional complication of our approach is that the direction that sensed data can be

forwarded toward the sink could be restricted by the interference due to scheduled transmissions. Hence, we need to incorporate some notion of interference's impact during the schedule construction. This is accomplished by enforcing a very mild requirement we call the *reachability constraint*. It essentially states that no interference from scheduled transmissions is possible to disrupt the ability of any node in the network to have at least one path from itself to the sink. Note that this property does not imply any optimality with respect to such a path. However, as the reader may have noticed, long paths are not fundamentally against our objectives because we are ready to accept increased latency in favor of higher throughput.

Definition 5.3.7. *A schedule S respects the Reachability Constraint if there exists at least one directed path $P_{v \rightsquigarrow s}$ from each node v to the sink node s where each intermediate node in sequence of the path can receive the transmission without collisions.*

5.4 Pipeline Scheduling Algorithm

We have now all the elements necessary to propose a solution. The schedule is to be created without paying attention to precedence constraints, but by merely ensuring that there exists reachability between all nodes and the sink. The purpose is to create a short schedule, which translates to high throughput. Once the schedule is completed, we produce the aggregation tree (a guarantee exists that at least one such tree exists due to reachability being satisfied) that exhibits the smallest possible collection latency cost. The described approach represents a radical departure from previous solutions. Algorithm 4 shows our approach.

Initially, the input graph must be transformed into a directed graph, with two arcs per edge. Another preliminary action is the removal of all outgoing arcs from the sink, because the sink does not transmit. These actions are executed by the function *TransfGraph* in Line 1.

Line 2 selects an order by which the vertices will be processed. We consider three heuristics. Each heuristic may employ multiple decision criteria. The first heuristic applies tree criteria. The first criterion, (a), is to select the vertex whose outgoing arc has the largest number of common neighbors between itself and the destination vertex. The rationale behind this heuristic is to identify the vertices inside large cliques on the graph and schedule them sooner, because they could remove more conflicting arcs (explained latter). As more than one vertex may have the same number of common neighbors, the second criterion, (b), is to give preference to vertices further away (hop distance) from the sink. The rationale is

Algorithm 4: Pipelined Aggregation Convergecast

Input: $G(V, E)$, $sink$
Output: $T, sched$

1: $G'(V, A) \leftarrow TransfGraph(G)$
2: $V_o \leftarrow OrderVertices(G')$
3: **for** ($i = 1$ *to* $|V_o|$) **do**
4: $\quad reachable \leftarrow$ **false**
5: $\quad sched(v_i) \leftarrow 0$
6: $\quad$ **while** ($reachable =$ **false**) **do**
7: $\quad\quad reachable \leftarrow$ **true**
8: $\quad\quad sched(v_i) \leftarrow sched(v_i) + 1$
9: $\quad\quad A^c \leftarrow ConflictArcs(G', sched, v_i)$
10: $\quad\quad reachable \leftarrow Reachability(G', A^c, v_i)$
11: $\quad\quad$ **if** ($reachable =$ **true**) **then**
12: $\quad\quad\quad G' \leftarrow G'(V, A \backslash A^c)$
13: $\quad\quad$ **end if**
14: $\quad$ **end while**
15: **end for**
16: **for** ($j = 1$ *to* $|A(G')|$) **do**
17: $\quad w[arc_j] \leftarrow SchedDifference(arc_j)$
18: **end for**
19: $G^* \leftarrow G'(V, A^T)$
20: $T_{MCA} \leftarrow MinCostArborescence(G^*, w, sched)$
21: $T \leftarrow (T_{MCA})^T$

to remove more arcs far from the sink first and leave the region close to the sink (where all paths must inescapably converge) with more path options. If still there is a tie, the third criterion, (c), selects first vertices of higher ID. The second heuristic is a variation of the first one whereby the criteria of hop distance from the sink is applied first (criterion (b)), and then the criterion of the number of common neighbors (criterion (a)), and then, in the event of a tie, the order is decided based on higher ID (criterion (c)). Finally, the last heuristic selects randomly a vertex order.

Once a vertex order is defined, the algorithm processes nodes following the order selected, trying to assign each vertex to transmit at the earliest possible timeslot. This is executed in lines 3 to 8. A vertex v_i is taken from the queue to be processed. A tentative schedule for node v_i is stored in $sched(v_i)$, representing the slot index within the schedule cycle. The boolean variable *reachable* is used to indicate if the sink node is reachable by all vertices on the graph.

Once a tentative schedule (i.e., slot in which to transmit) is selected for vertex v_i, it is

time to verify if this selection does not break the reachability restriction. First, the algorithm determines which are the conflicting arcs. Conflicting arcs are those that would not be possible to be used because their activation would violate the primary or secondary interference constraints. The conflicting arcs are determined by the function *ConflictArcs* in Line 9. Clearly, the interference is applied here to restrict the paths available to reach the sink instead of restricting the time slots available for vertex transmission.

Line 10 is used to verify if the removal of the conflicting arcs, identified in Line 9, would break the reachability constraint. Function *Reachability* checks if the sink node is reachable from each v_i's 1-hop and 2-hop neighbors.

Let $\Gamma_1(v_i)$ be the set of v_i's one-hop neighbours, and $\Gamma_2(v_i)$ the set of two-hop neighbours. Let the set of nodes in v_i's local region (inclusive of v_i) be defined as $\mathcal{N}(v_i) = \{v_i \cup \Gamma_1(v_i) \cup \Gamma_2(v_i)\}$.

Definition 5.4.1. *Convergecast Reachability is the property according to which all vertices $u \in V$ in the directed graph $G'(V, A)$ can reach the sink node.*

Lemma 1. *A directed graph G', which satisfies the convergecast reachability, maintains this property when vertex v_i is additionally scheduled,* if and only if *all $w \in \mathcal{N}(v_i)$ can reach the sink after v_i is scheduled. This can be trivially verified by inspecting the reachability of the nodes in $\mathcal{N}(v_i)$.*

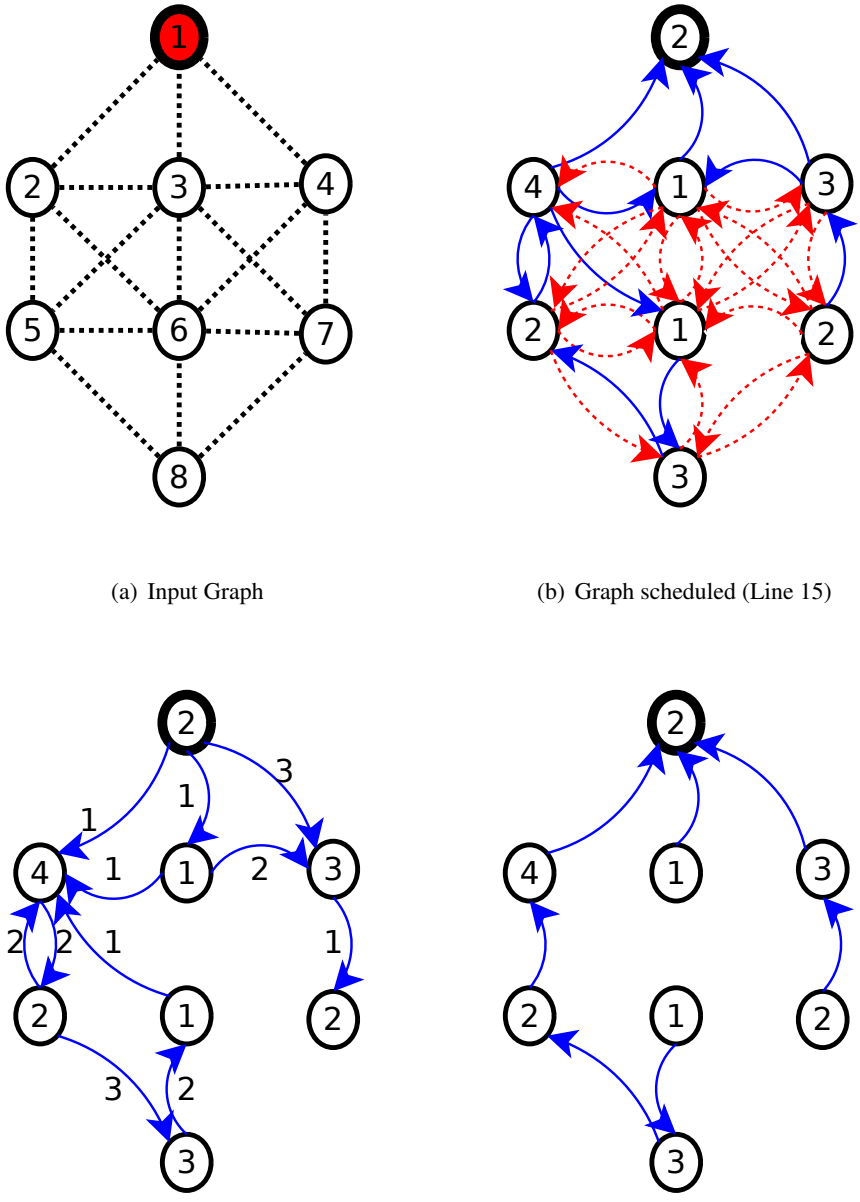
Proof. A vertex transmission only affects incoming arcs to the vertices in $\{v_i \cup \Gamma_1(v_i)\}$, because these are the vertices affected by the primary and secondary conflicts. An incoming arc to $\Gamma_1(v_i)$ comes from at most vertices in $\Gamma_2(v_i)$. An interruption in a directed path $P_u = \{\overrightarrow{uw}, \overrightarrow{w, z_1}, \overrightarrow{z_1, z_2}, \cdots, \overrightarrow{z_k, sink}\}$ (starting in vertex u) also interrupts all directed paths where P_u is a subset. Only vertices for which all their possible paths to the sink cross vertices in $\mathcal{N}(v_i)$ may have their reachability affected, as they would otherwise have an alternative path that does not involve $\mathcal{N}(v_i)$. Assuming that at least one vertex of its path is in $\mathcal{N}(v_i)$, then checking if all vertices in $\mathcal{N}(v_i)$ maintain their reachability to the sink is sufficient to guarantee that the remaining vertices in V also preserve their reachability to the sink. The contrary is also true, because $\mathcal{N}(v_i) \subset V$. Therefore, a directed graph G' maintains the convergecast reachability property by inspecting only whether the vertices in $\mathcal{N}(v_i)$ maintain their reachability to the sink subject to v_i's schedule. □

The reachability verification can be performed using Breadth First Search (BFS), based on Lemma 1. Each vertex in $\mathcal{N}(v_i)$ is selected as root. If no reachability violation occurs, the tentative schedule is valid, and is committed. If the reachability restriction still holds, all the conflicting arcs A^c, identified before, are removed from graph G'. The removal is executed in Lines 11 to 13. If a reachability violation happens, no arc is removed, and the same vertex v_i is processed again with a new tentative schedule (at the next slot). If the search for a schedule slot for v_i without producing reachability violation turns up fruitless, i.e., all slots thus far defined for the schedule have been exhausted, then the schedule length expands by one slot (by virtue of Line 8) and vertex v_i is trivially scheduled for transmission in that slot.

Once a schedule is defined for all vertices, we need the aggregation tree, preferably one that minimizes the delay. The intermediate result so far is a directed subgraph with the transmission time for each vertex. Each vertex might have more than one path to reach the sink node. After node v_i transmission, the information content is only forwarded further to the sink after node v_i's destination transmission. This slack time between the source vertex u transmission and the destination node v transmission can be used to differentiate between two or more possible candidates, say v' and v'', i.e., whether to use arc $\overrightarrow{uv'}$ or arc $\overrightarrow{uv''}$. Therefore, we calculate the slack times for all nodes in Lines 16-18 and use them as the weight of each arc. If the destination vertex is scheduled after the source vertex, the time difference will be $w[\overrightarrow{uv}] = sched(v) - sched(u)$. If vertex v is scheduled before vertex u, we make use of the fact that the information collection is periodic, and that the schedule is periodically repeated. If the destination vertex has already transmitted in the current cycle, the time slack lasts until v's transmission on the next cycle. Therefore, the time slack will be $w[\overrightarrow{uv}] = max(sched) - sched(u) + sched(v) + 1$. Here $max(sched)$ indicates the length of the schedule cycle (in slots).

Each arc has now a weight expressing the time elapsed from the arc's source transmission to the time it is forwarded by the arc's destination (or the delay associated to the "use" of this arc). Our objective now is to find an aggregation tree that minimizes the overall delay. How can such minimum weight tree be obtained? The traditional algorithms (*Kruskal* and *Prim* [32]) to obtain a spanning tree T use an undirected graphs. Therefore these algorithms cannot be directly used. The problem is related to computing a rooted directed spanning tree. The rooted directed spanning tree is a graph which connects, without any cycle, all nodes with $n - 1$ arcs (each node except the root) and each node has one and only one

incoming arc. This formulation belongs to a class of *branching* problems, also known as *minimum cost arborescence* (MCA) [10]. An algorithm for solving this problem has been proposed by Edmonds [39]. The MCA algorithm is capable of obtaining a result even if the input graph has cycles.



(a) Input Graph

(b) Graph scheduled (Line 15)

(c) MCA input (Line 19)

(d) Final result (Line 21)

Figure 5.2: Execution of Algorithm 4

More specifically, convergecast can be seen as an *in-branching* problem [10]. Therefore,

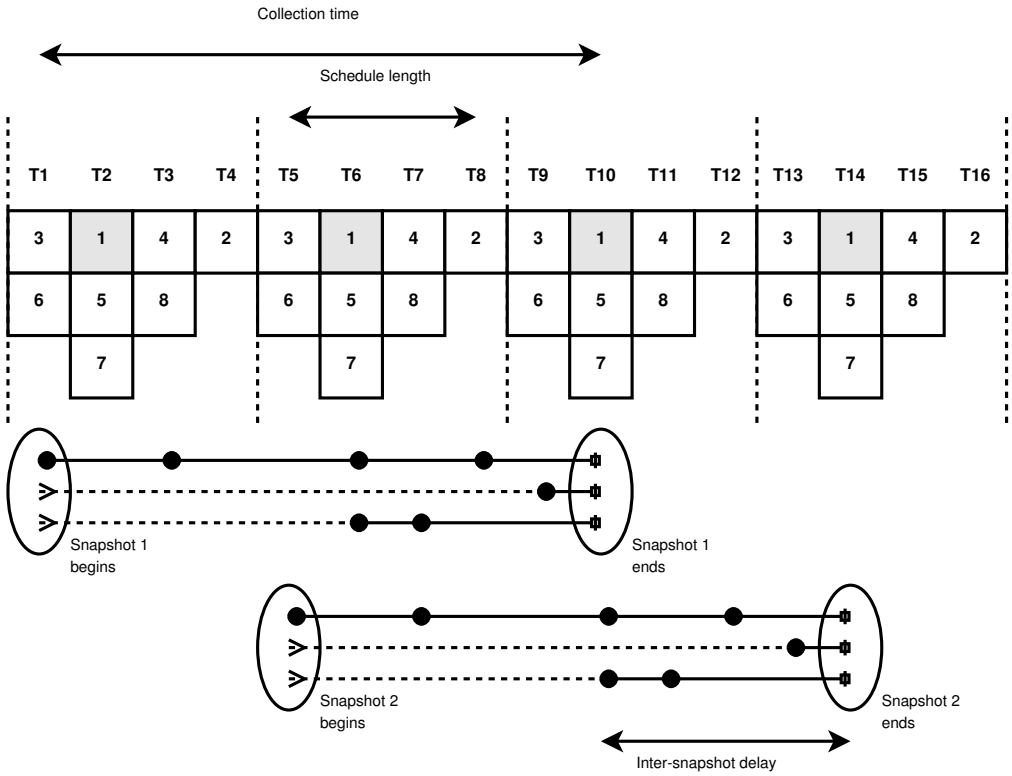


Figure 5.3: Pipeline of Algorithm 4

a slight modification on Edmonds' algorithm is enough to obtain an aggregation tree. The modification consists in changing the direction of the arcs obtained after Line 18, using the same weights obtained before. This operation is executed in Line 19. In Line 20, the minimum cost arborescence algorithm is used and an outward tree is obtained. The last step is executed in Line 21, and consists in reverting back the arc's direction. In the end, we have the schedule obtained in Line 15, and the aggregation tree obtained on Line 21.

An example of execution of the algorithm is presented on Figure 5.2. Figure 5.2(a) contain the input graph with node IDs. The result of the first phase is depicted in Figure 5.2(b). The number inside the circles indicates the timeslot selected for each node transmission. The dotted arcs indicate the conflicting arcs removed during the schedule selection. Figure 5.2(c) represents the status after execution of Line 19, where the directed graph is ready to be used by MCA algorithm. The number besides each arc indicates the time slack. The final result of the pipeline aggregation convergecast algorithm is depicted on Figure 5.2(d). The aggregation convergecast execution is depicted in Figure 5.3. In this example, one snapshot propagation may span three schedule cycles (only two snapshots are shown in the Figure), therefore, up to three snapshots are propagating at the same time through the network.

The box of the sink node is depicted in gray because it does not actually transmit. Sink's presence on Figure 5.3 is provided to delineate when a snapshot is completely received.

5.4.1 Complexity Analysis

The run-time complexity of *Pipelined Aggregation Convergecast* is as follows: **Line 1** has run-time complexity of $\mathcal{O}(|V|)$. **Line 2** may have different run-time complexity depending of the heuristic used. The first heuristic *ACSPIPE_1* is the more demanding, because it has to transverse each edge of the graph and get the first and second neighborhood of each of its endpoints in order to define how many common neighbors the endpoints have. Therefore, its run-time complexity is $\mathcal{O}(|E||V|^2)$. The second heuristic *ACSPIPE_2* is executed in $\mathcal{O}(|E| + |V|)$ because a BFS is enough to define which layer each vertex belongs to. The last heuristic *ACSPIPE_3* needs a run-time complexity of no more than $\mathcal{O}(|V|)$ to get a random vertex. The vertex ordering, using weights from the heuristics, can be executed in $\mathcal{O}(|V|^2)$. The execution of *ACSPIPE_1* heuristic represents the worst case. Therefore, the run-time complexity of **Line 2** is $\mathcal{O}(|E||V|^2)$.

The scheduling part is executed from **Line 3** to **Line 15**. **Line 3** shows that each vertex must be picked to be scheduled Thus, it is executed $|V|$ times. **Line 6** indicates that each vertex may be rescheduled n times, until there is no conflict with the previous scheduled vertices. The estimation of the magnitude of n is hard. However, we know that $n \leq |V|$, because $|V|$ is the schedule length's upper bound, or the maximum number of timeslots to be tested on each vertex. **Line 9** obtains the conflicting arcs surrounding vertex v. It is possible to discover the conflicts inspecting v's first and second neighbors and verifying their scheduled time. Therefore, it is necessary at most $|V|^2$ steps. The reachability verification, executed in **Line 10**, can be done by BFS on $\mathcal{O}(|E| + |V|)$. **Line 12** is a simple arc removal, which has a run-time complexity of $\mathcal{O}(|E|)$. The remaining lines of the scheduling part are executed with run-time complexity of $\mathcal{O}(1)$. The worst case run-time complexity of the scheduling part is $\mathcal{O}(|V|^4)$.

The algorithm last block defines the routing part. The slack time, calculated in **Line 16** and **Line 17**, has run-time complexity of $\mathcal{O}(|E|)$. **Line 19** and **Line 21** are simple arc inversions, each one executed in $\mathcal{O}(|E|)$. The analysis of **Line 20** involves the run-time complexity of Edmonds' algorithm, used to obtain the minimum cost arborescence. According to Tofigh [104], Tarjan described an implementation of Edmonds' algorithm in [102] that runs in $\mathcal{O}(|E|log|V|)$. With a simple modification, the algorithm can run in $\mathcal{O}(|V|^2)$, which is

more suitable for dense graphs. An implementation error is corrected by Camerini *et al.*in [20]. Gabow *et al.*[46] give an $\mathcal{O}(|V|log|V| + |E|)$ implementation for optimum spanning arborescence. The authors of [46] note that is not possible to improve on the time complexity for any Edmonds' algorithm implementation because the algorithm can also be used to sort n numbers, and sorting n numbers requires $\mathcal{O}(n \log n)$ time. Since it always has to inspect every edge of the graph, we cannot expect to find a better run-time complexity for Edmonds' algorithm than $\mathcal{O}(|V|log|V|+|E|)$. Therefore, we will assume that the run-time complexity for Edmonds' algorithm is $\mathcal{O}(|V|log|V| + |E|)$.

Based on the analysis of the 3 parts of the *Pipelined Aggregation Convergecast* algorithm, setup and vertex order definition (**Lines 1-2**), scheduling part (**Lines 3-15**), and routing part (**Lines 16-21**), the run-time complexity of our algorithm is $\mathcal{O}(|V|^4)$.

5.5 Experiments

We evaluate our algorithm performance using a set of connected graphs generated by placing sensors in a square region of size 200×200. The sensor positions are uniformly randomly distributed over the area. The sink node position is also random. Each node has transmission range of $R = 25$, unless otherwise noted. We used sets of nodes ranging from 200 to 800 nodes. Each point in the figures represents the average value for a set of ten graphs with the same number of sensors. The error bars represent 95% confidence intervals.

We implemented the three version of node ordering described in Section 5.4. The first algorithm *ACSPIPE_1* represents the ordering by common neighbor count first. *ACSPIPE_2* indicates the ordering by farthest layer first, and *ACSPIPE_3* is the random order.

We use *WIRES* [76] and *BDMRST* [52] algorithm for comparison. *WIRES* represents a solution using aggregation convergecast without pipelining where all the precedence constraints are satisfied without a single schedule cycle, and it is designed using the traditional two phase approach. *BDMRST* uses pipelining, but pre-selects an aggregation tree with some characteristics. As BDMRST algorithm uses a multi-channel scheduler, we limit the algorithm to a single channel.

Figure 5.4 shows the schedule length produced by each algorithm, while Figure 5.5 presents the aggregate throughput. The collection delay is portrayed in Figure 5.6. The aggregation

tree produced by each algorithm is characterized in Figure 5.7 by means of the maximum node in-degree.

Figure 5.8 exhibits the tradeoff between throughput and delay by each algorithm. Similarly, Figure 5.9 shows the relation between the schedule length and the collection delay. The points on both graphs are average values of sets with the same number of nodes.

5.5.1 Discussion

It has been repeatedly observed that, in general, scheduling data transmissions on wireless networks reveals tradeoffs between throughput and delay [79, 105]. The situation is no different with aggregation convergecast. The impact of trying to minimize latency is clearly seen in Figure 5.6, where WIRES exhibits the lowest data collection latency because of the linkage between delay and throughput (one is the reciprocal of the other). This linkage results in limited aggregate throughput, as shown in Figure 5.5. The removal of emphasis from latency by means of allowing the precedence constraints to be satisfied over multiple cycles expands the solution space. ACSPIPE and BDMRST can explore a larger solution space and new tradeoff possibilities. The aggregate throughput improves, but collection delay is penalized.

Networks with large number of nodes may have an additional limitation to achieve better aggregated throughput. If the transmission range is fixed, the throughput is limited by two factors: (a) in smaller and dense areas, the interference will constitute the major influence to restrict a smaller schedule length, (b) for wider and sparse areas, with uniform node dispersion, the number of hops from each node to the sink is the most influential factor on the increase of the schedule length. In both cases, the number of time slots necessary tends to be large, and the overall throughput will decrease. In both cases, the use of pipelining may be beneficial, because more than one snapshot may propagate through the network per time.

Precedence constraints are not the only element to restrict the solution space in our experiments. The pre-selection of an aggregation tree with some predefined characteristics by BDMRST's (bounded-degree and minimum radius) also restricts the solution space. The effect of bounded-degree is observed on Figure 5.7. The end result is a solution superior to WIRES in aggregate throughput, but inferior to ACSPIPE. Besides, BDMRST's collection latency is not significantly better than ACSPIPE's (Figure 5.6). It is fair to say that

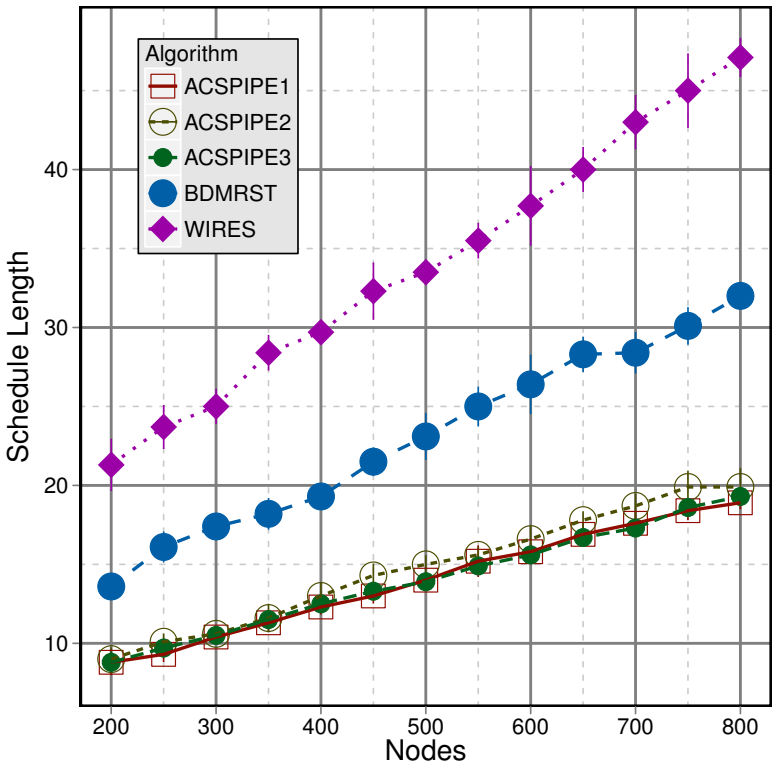


Figure 5.4: Schedule Length

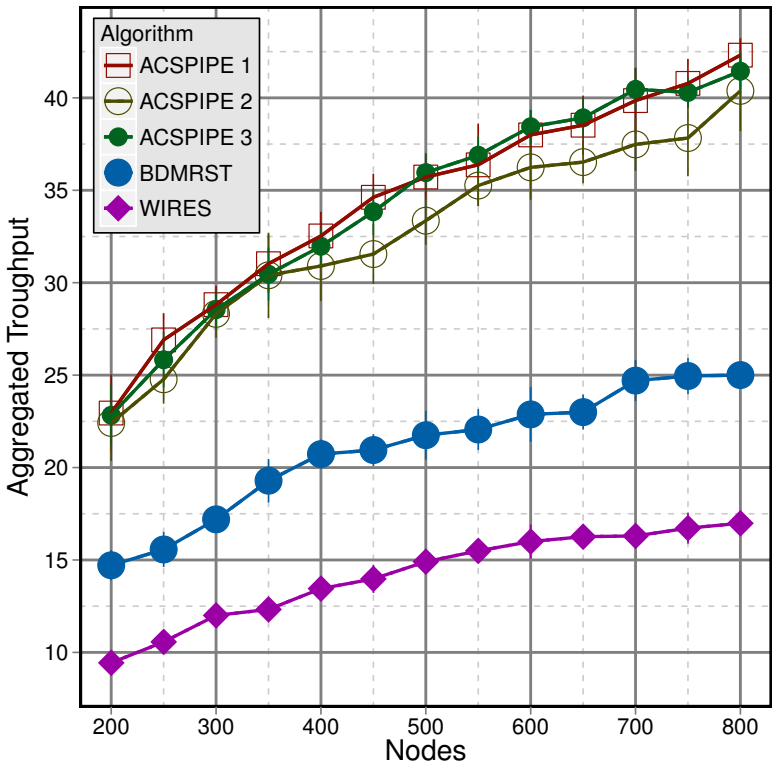


Figure 5.5: Aggregate Throughput

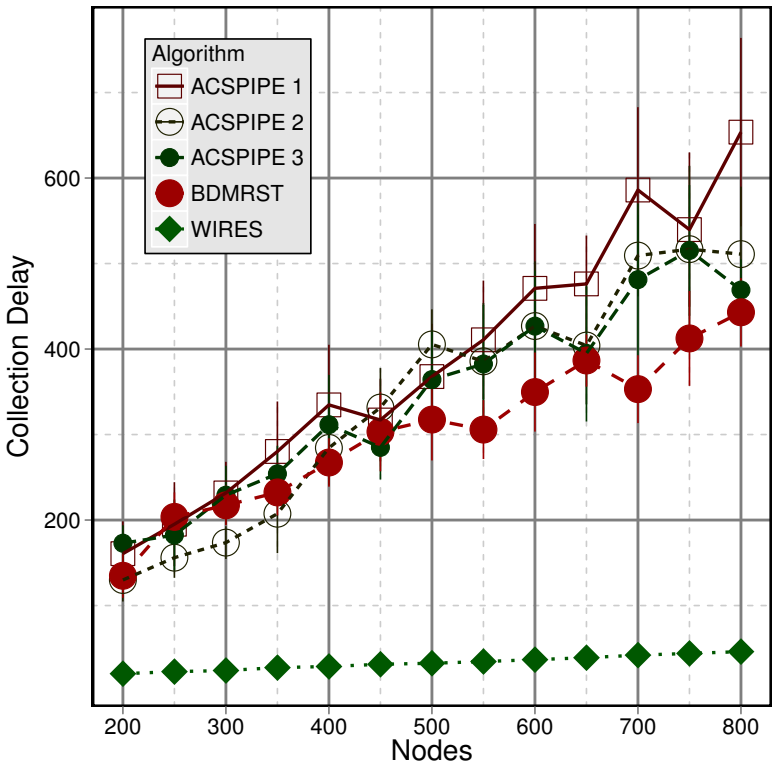


Figure 5.6: Snapshot Collection Delay

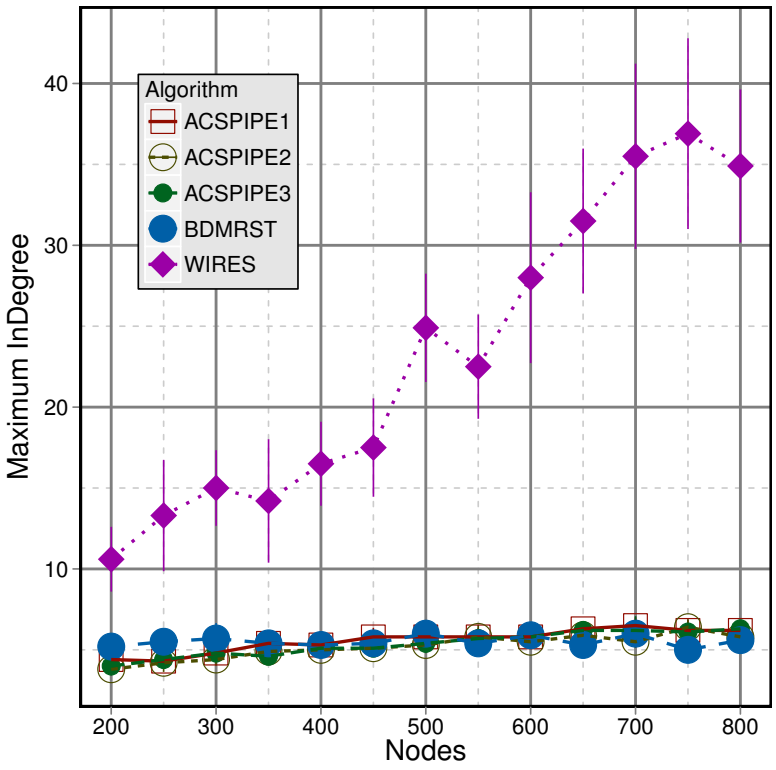


Figure 5.7: Maximum In-Degree

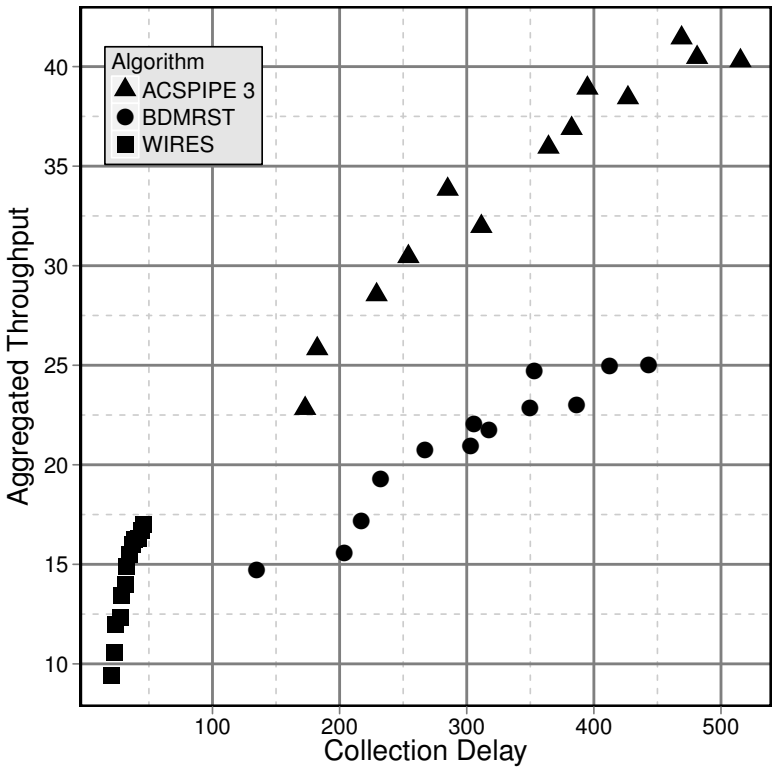


Figure 5.8: Throughput vs. Delay

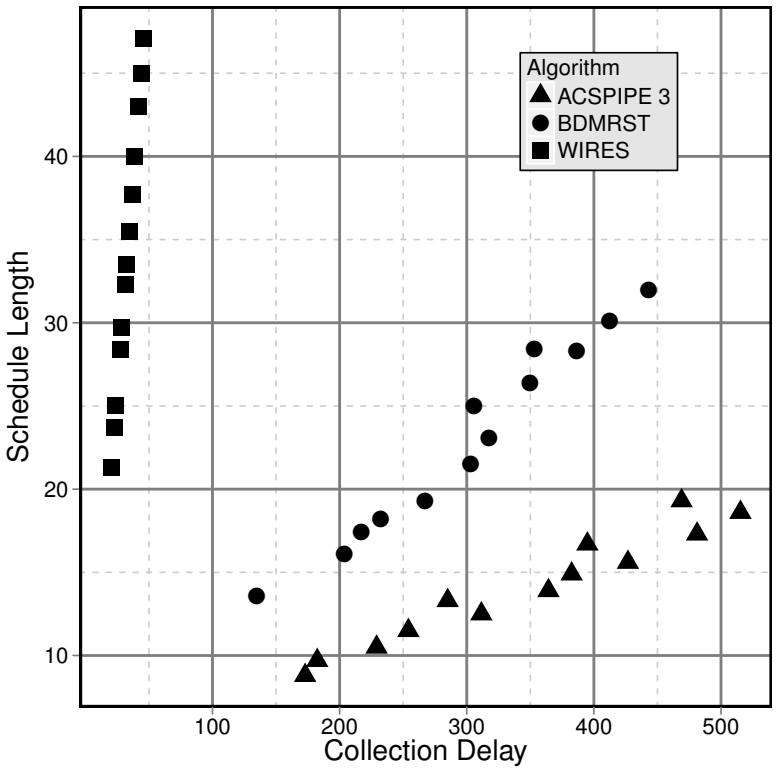


Figure 5.9: Schedule Length vs. Delay

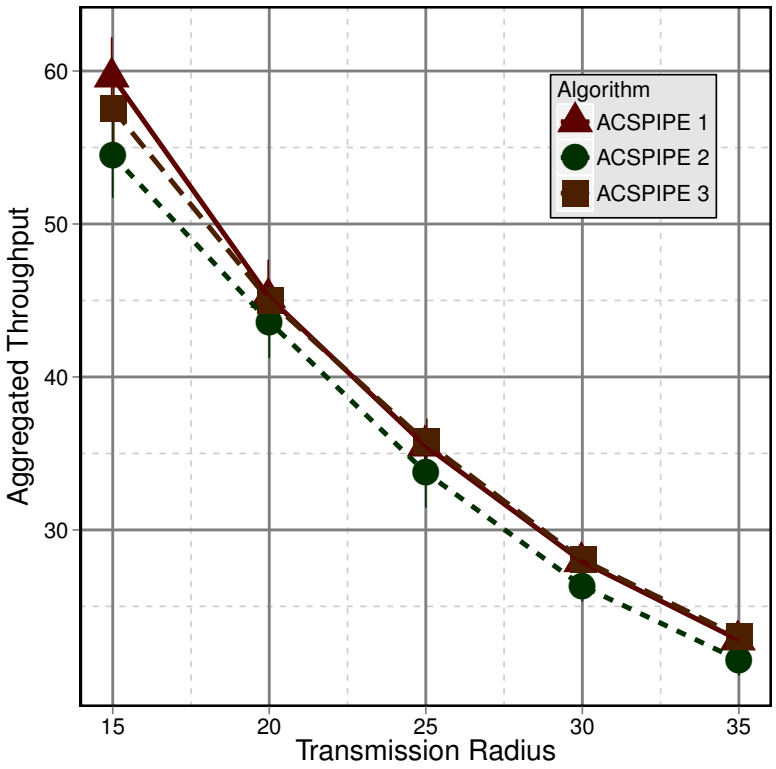


Figure 5.10: 500 Nodes: Throughput vs. TX Range

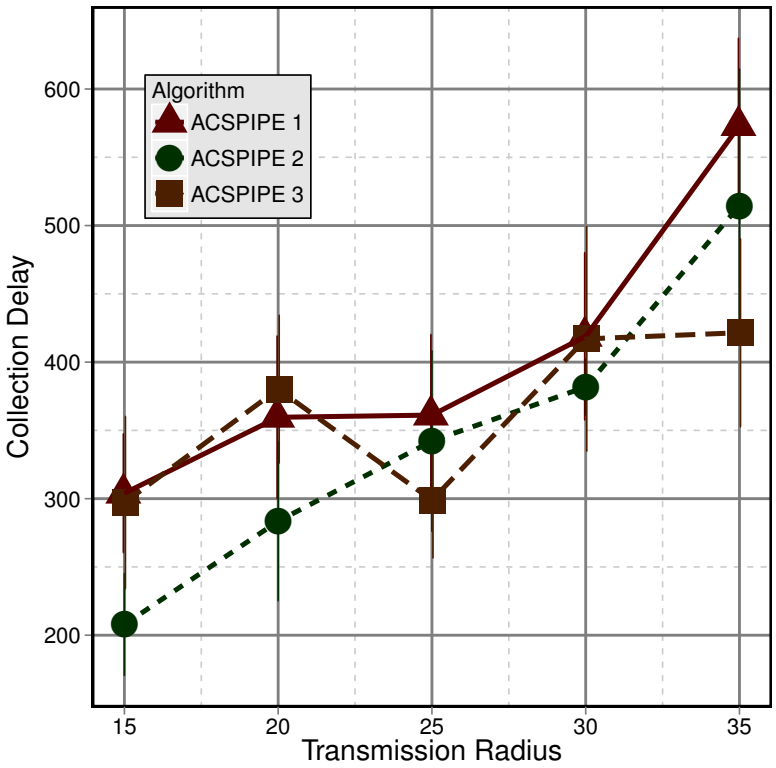


Figure 5.11: 500 Nodes: Delay vs. TX Range

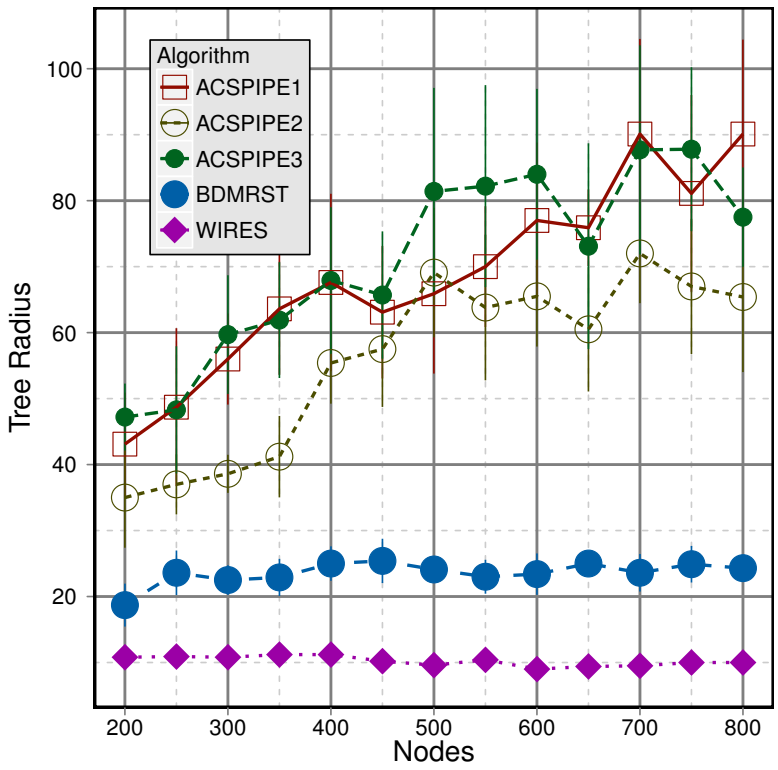


Figure 5.12: Tree Radius

the intention of BDMRST's authors is to use their solution in a multi-channel environment, with enough frequencies to eliminate secondary conflicts. Therefore, the only obstacle to improve schedule length is the tree radius (in which BDMRST shows better results than ACSPIPE). If this condition (existence of enough frequencies to eliminate all secondary conflicts) is not met, clearly BDMRST is limited, because it restricts the potential aggregate throughput, without substantial gain in the collection latency.

An interesting behavior to notice among ACSPIPE heuristics is that the second heuristic (that gives priority to vertices further away from the sink) presents a slightly different result. After 350 nodes, the aggregation throughput (Figure 5.5) is inferior than the other two. This fact is also reflected in the tree radius (Figure 5.12), where the tree depth is smaller. Even with a shorter tree, the collection delay is not much better. The observation suggests that the longer the tree radius, the smaller is the pipelining schedule. WIRES *straight line* shape shown in Figure 5.9 is consequence of the previously proposed schemes, where $\Delta_c = l$.

We also observed that ACSPIPE presents a natural decrease of aggregate throughput when the transmission radius increases (Figure 5.10). A larger transmission range increases the

interference, consequently more time slots are necessary to overcome the contention, and the collection delay will consequently increase (Figure 5.11).

5.5.2 Optimal Solution for Small Networks

We also evaluate the results produced by our algorithm against the optimal solution for small networks. As the problem has two main criteria (schedule length and collection delay) we search for a optimal solution that has, at first, the smallest possible schedule length, and latter, among the solutions with the smallest schedule length, we selected the solutions with the smallest collection delay. The result is described in Table 5.1.

The experiments used 10-node graphs, with a variable number of links (listed in the second column of Table 5.1). The remaining columns describe the results for different circumstances, initially for the optimal solution (*Optimal*), then solutions using the first heuristic for node order (*ACSPIPE_1*), and, finally, solutions using the second heuristic node order (*ACSPIPE_2*). The last column lists the time spent on full search of the optimal solution. Each algorithm is divided into two columns *SCH* and *DLY*, representing schedule length and collection delay respectively. Both columns represent timeslot units.

The optimal results were generated using the Algorithm 4, but trying all $V!$ possible node orders. The results shows that heuristic *ACSPIPE_1* and *ACSPIPE_2* produce schedule lengths very close to the optimal. However, distance between collection delay of the optimal solution and collection delay using the proposed heuristics suggests that there are some room for improvement.

Table 5.1: Pipelined Aggregation Convergecast Optimal Solutions

	Links	Optimal		ACSPIPE_1		ACSPIPE_2		T(sec)
		SCH	DLY	SCH	DLY	SCH	DLY	
S0	31	7	7	7	14	7	7	909
S1	26	6	6	6	12	6	11	817
S2	24	4	6	5	11	5	10	660
S3	31	6	6	6	18	6	16	829
S4	27	5	6	5	10	6	11	713
S5	35	5	6	6	6	6	6	822
S6	21	4	6	5	9	4	12	517
S7	21	6	8	6	19	6	12	707
S8	16	4	5	4	15	4	11	417
S9	20	7	7	7	14	7	7	630

5.5.3 Energy Consumption

Wireless sensor networks have resource constraints. Nodes are usually battery powered, then energy storage becomes a major limitation for a long-lived sensor network operation. The main source of energy drain is packet transmission and reception. Therefore, an algorithm that produces routing and scheduling solution that requires lesser transceiver operations will preserve more energy. Even though data aggregation scheme is *per se* an efficient energy saving scheme, the number of transmissions and receptions per node continues to play a important role. As we are using data aggregation in all algorithms, this aspect will not be our focus. We want to know which algorithmic solution produces a logical topology and schedule that minimizes the energy use.

Several models capturing the energy consumption have been proposed [67, 106]. They captures several aspects of problem. However, we want a simpler model where the influence of the logical topology and schedule length over time are the only aspects present. We designed a simplified energy model presented next.

$$E(v_i, a, T) = deg(v_i, a) * \frac{T}{l(a)} \tag{5.1}$$

Equation 5.1 expresses the amount of energy units spent by the node v_i during T time slots, when the algorithm a is executed. The execution of algorithm a produces a solution where node v_i has degree $deg(v_i, a)$ in the logical topology, and the schedule length is $l(a)$ time slots on each cycle. The energy used by node v_i reflects the influences of logical topology and schedule length. The interpretation is straightforward. Logical topologies with nodes having smaller degrees may use less energy units for transmission and reception, because there will be less receptions to be executed by the transceiver. However, a topology with low-degree nodes may end up spending more energy units than a high-degree topology during a fixed amount of time T if it is used more frequently (schedule part). The reason why it might happen is because of a smaller schedule length. A smaller cycle requires that a node transmits and receives packets more often, consequently using more energy.

Nodes with the smallest degrees are those who use the smallest amount of energy. They are leaves in the logical topology. Leaves only spend energy for its transmission, because they have no children to spend energy in the reception. The amount of energy consumed by them over time is only dependent of the schedule length. By other hand, nodes with the highest energy consumption are those with the highest degree.

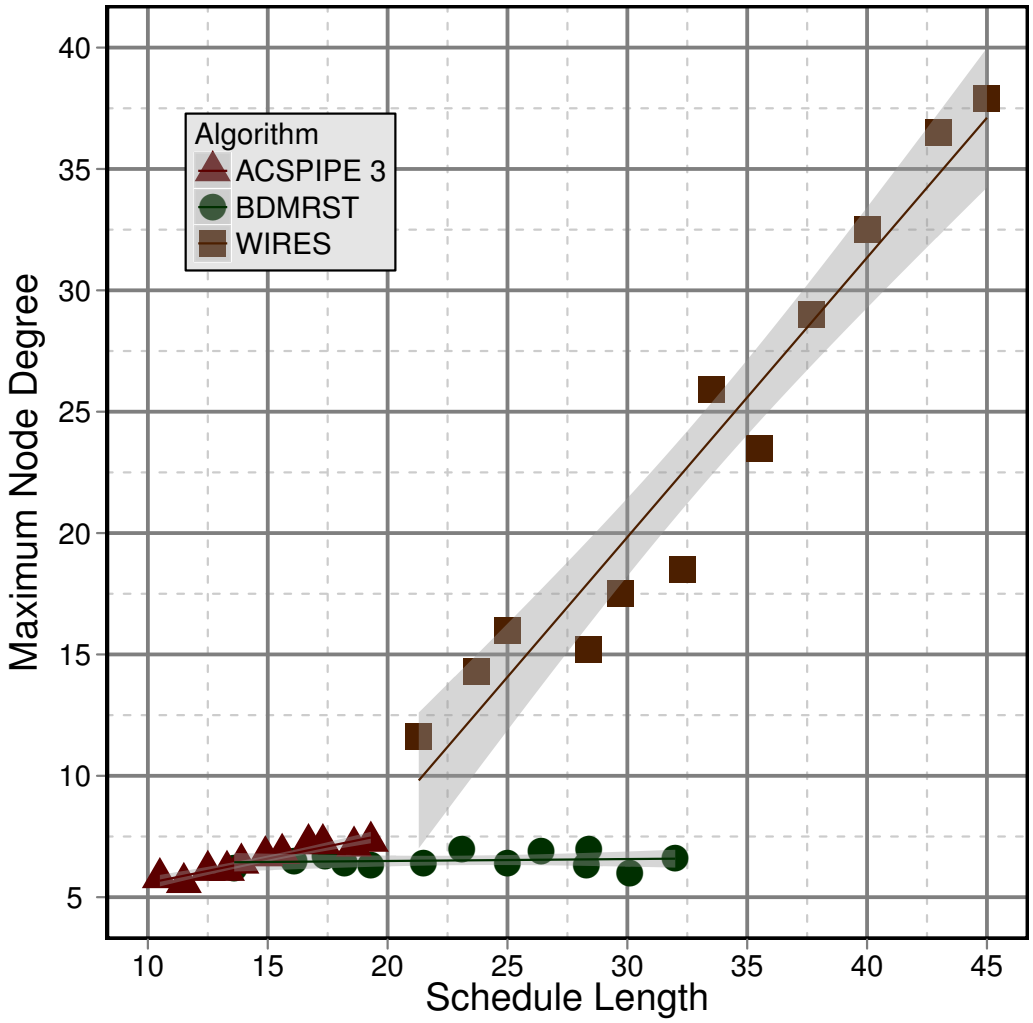


Figure 5.13: Relationship between Schedule and Maximum Node Degree

We are interested in the rate by which energy is spent over time by each algorithm. According to the energy model described in Equation 5.1, this rate is given by the ratio between node degree (logical topology) and the maximum schedule length. For the sake of comparison, we use nodes with the largest degree, because they express the energy consumption rate that a node may have. The results are expressed on Figures 5.13 and 5.14.

Figure 5.13 presents the relationship between the two factors that influence node's energy consumption. Each point in the graph represents the average value of schedule length and maximum node degree. The lines on each algorithm group is a linear regression of the average values, and the shade represents the confidence interval of 95%. Figure 5.14 shows the energy consumption rate for the maximum degree nodes on each algorithm, for different

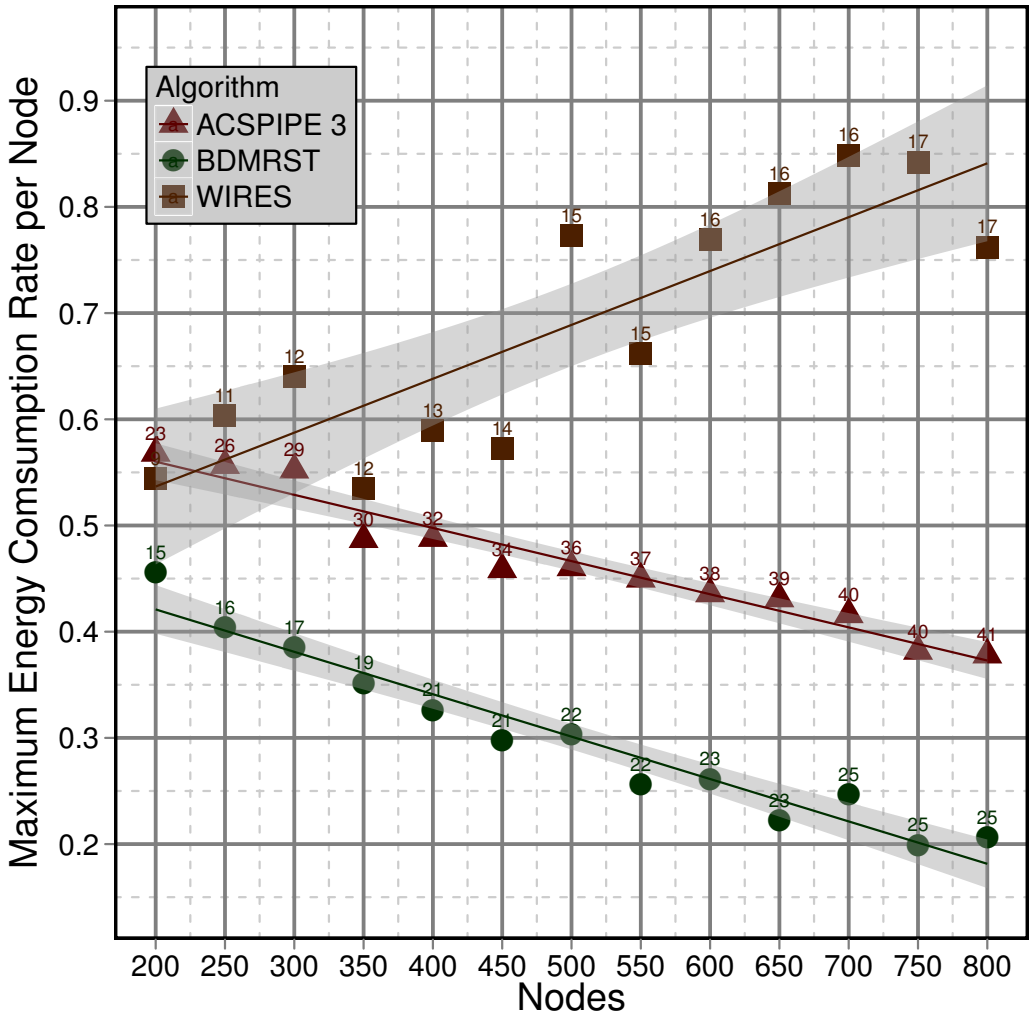


Figure 5.14: Energy Consumption Rate According to Node Density

node densities. The number near to each point is the average throughput achieved by the algorithm on the particular node density.

The BDMRST algorithm has a constant maximum node degree (as designed), and the lowest energy consumption rate among all protocols. Despite all that, the throughput is lower than ACSPIPE. Our algorithm closely follows the topological characteristics of BDMRST, with small maximum degree. This is a welcomed characteristic. This property is not explicitly encoded in the algorithm, as it is the case with BDMRST. In our case, it comes as an intrinsic and attractive feature. ACSPIPE energy consumption rate is higher than BDMRST because it is able to achieve higher throughput rate. Both protocols (BDMRST and ACSPIPE) decrease their energy consumption rate as the node density increases. WIRES

presents a completely different dynamic. The selection of a logical topology that exclusively addresses the influence of precedence constraints on the solution produces a different relation between node degree and schedule length. The result is seen on Figure 5.13. The final consequence is an increase on the energy consumption rate as node density increases. For WIRES, the yielded logical topology (SPT) plays a big role on the energy consumption results. Its emphasis in minimizing precedence constraints may also lead to longer schedule lengths [99].

The use of pipeline definitely affects the energy rate required to execute a solution for the aggregation convergecast problem. The algorithm we propose produces low node degree topologies. Even though the energy consumption rate might be higher than other algorithm, the rate is higher due to a higher throughput achieved, therefore, more node activity per time.

5.6 Conclusion

This Chapter examines the influence of pipelining on the solution of the aggregation convergecast problem. By expanding the satisfaction of precedence constraints over multiple schedule cycles and by transferring multiple data collection snapshots in parallel, we are able to attain higher throughput which is necessary for certain classes of sensor network applications.

We propose a different approach to account for interference during the schedule construction phase. Specifically, it is used to account for the possibility that it restricts the paths available from nodes to the sink but without committing to a particular spanning (aggregation) tree. This is in sharp contrast to using interference with a given aggregation tree to limit the time slots when a node is allowed to transmit. Essentially, the only limitation for a node to be allowed to transmit in a time slot is to ensure that it does not preclude the existence of a directed path connecting some other node to the sink. We call this restriction *reachability constraint*.

We designed a new algorithm that uses pipelining and is based on preserving reachability during schedule construction. We compared it with two other algorithms, one from the traditional two-phase (routing first, scheduling second) variety and another that uses pipelining but with a pre-defined aggregation tree. Even though the proposed algorithm is not providing the optimal throughput, our approach is able to present solutions with high throughput,

albeit at the cost of latency.

Under a different light, our work is nothing more but another expression of the well–known tradeoff of throughput versus latency. It has been recognized that for fixed random networks, higher throughput can only be obtained at the cost of increasing delay [47]. In this respect, the novelty of our contribution is in demonstrating how to structure the solution space for aggregation convergecast scheduling such that the interplay of throughput versus delay can be captured.

Chapter 6

Conclusion and Future Work

In this Thesis we address the aggregation convergecast problem, when non-conservative flows and precedence constraints are present, and when a pipelining approach is used. We start creating a constraint programming model of the aggregation convergecast problem. We reveal that, most of the time, the optimal aggregation tree has tree size greater then a shortest path tree [99]. This revelation induces the need to search for an aggregation tree that better captures the nature of the restrictions acting in the problem. We propose that the aggregation tree should be the combination of a shortest path tree, which minimizes the precedence constraints, with a minimum interference tree, which minimizes the resource constraints [98]. Together, both trees form a shallow light tree, which incorporates and balances both constraints. After obtaining this new tree, the next step is the selection of a feasible and minimum schedule, using the aggregation tree obtained. We show that the process to acquire the minimum schedule of a (directed) aggregation tree is a NP-Complete problem, similar to Mixed Graph Coloring. We propose a mapping from the aggregation convergecast problem to the mixed graph coloring problem. We named it extended conflict graph. The extended conflict graph captures both precedence and resource constraints. The extended conflict graph allows the search for the optimal solution using a branch-and-bound algorithm. The results show that smaller schedules are indeed possible, if shallow light trees are used.

Next, we take the risk of departure from the traditional requirements of aggregation convergecast and drop the requirement that a data aggregation collection must be completed in only one schedule cycle [100]. The departure is done exploring the use of pipeline in the scheduling. Not only we remove the one period requirement, but also we invert the long held approach of obtaining first the aggregation tree and latter the schedule. We investi-

gate the possibility of selecting the schedule for the problem first and latter choosing the aggregation tree. This inversion is possible because we introduce the notion of reachability constraints, such that a schedule is only accepted if it respects the reachability constraint, in other words, if each node can reach the sink node by at least one path.

6.1 Contributions

The most important contributions of this research are detailed below:

- ✓ **Aggregation Convergecast CP Model**

  We model aggregation convergecast as a Constraint Satisfaction Problem to obtain the shortest possible schedule for a TDMA frame that allows the complete data collection to the sink in single period, without interference. We use constraints related to topology, interference, and application logic, as well as their relationships. Our model is efficient compared to other model implementations for wireless networks available in the literature [21, 91].

- ✓ **Insufficiency of Precedence Constraints as Unique Criterion for Optimal Aggregation Trees**

  We observe that aggregation trees obtained exclusively by the minimization of precedence constraint criterion (like SPT) is very likely not the best choice for an aggregation tree. While computational limitations do not allow us to derive results for large networks, the results for smaller networks provide us valuable insights about the missing requirements to construct the optimal schedules and corresponding aggregation trees. The missing piece turn out to be the resource constraint criterion, originated from the wireless interference. Selecting non-SPT topologies, and avoiding the use of internal links of large cliques, have the potential to produce shorter schedules, improve the number of concurrent transmissions and enhance the throughput of the solution.

- ✓ **Algorithm to Balance Precedence and Resource Constraints in the Aggregation Tree**

  We explore in depth the use of precedence and resource constraints. Existing solutions prioritize one constraint over another instead of approaching the problem as a case of bi-criteria optimization. We propose a method to combine both constraints such that the resulting logical topology is a synthesis of the properties of a shortest

path tree (minimum precedence constraint tree) and properties of a minimum interference tree (minimum resource constraint tree). We derive the minimum interference tree from the idea of minimum cost arborescence, and use a interference measurement count as weight, while for the shortest path tree, we used a balanced shortest path tree algorithm. The merge of both trees is executed using a modified version of the *Light Approximate Shortest-Path Trees*, also known as *Shallow Light Tree*. The final result balances the contribution of each requirement, such that the final tree is closer to the optimal solution.

✓ **Relation of Aggregation Convergecast Scheduling and Mixed Graph Coloring**
We present the relation of Aggregation Convergecast Scheduling and Mixed Graph Coloring problems. This relation allows us to show the complexity of the Aggregation Convergecast Scheduling. If an aggregation tree is given, the scheduling is yet NP-Complete. Using this relation, previous common knowledge about Aggregation Convergecast can be formally shown.

✓ **Extended Conflict Graph Concept**
The conflict graph is a representation derived from the topology (communication) graph for the purpose of achieving a conflict-free schedule. The basic idea of the conflict graph representation is that every independent vertex set on the conflict graph can be scheduled simultaneously, i.e., in the same slot. Therefore, the coloring of a conflict graph defines a valid schedule. Unfortunately, this representation is insufficient to produce an independent vertex set on the conflict graph because precedence constraints have to also be satisfied. We widen the concept of a conflict graph and create an extended conflict graph, which encompasses transmission conflicts and precedence relations. Specifically, when a link activation (expressed by a node in the conflict graph) is required to be executed after another link activation (another node in the conflict graph), an arc is introduced in a mixed graph to express this precedence. Consequently, it is possible to represent the scheduling part of Aggregation convergecast, as a Mixed Graph Coloring problem.

✓ **Branch-and-Bound Algorithm for Aggregation Convergecast Scheduling**
We propose a branch-and-bound algorithm, based on an enumeration strategy, to obtain the optimal solution of Aggregation Convergecast Scheduling. Using the proposed algorithm, we demonstrate, through numerical results, that the convergecast aggregation tree, balancing precedence and resource constraints, can achieve better

quality results than the current state-of-the-art algorithms.

✓ **New Paradigm for the Two-Phase Approach in Aggregation Convergecast**
Traditionally, aggregation convergecast problem has being tackled by dividing the problem into two phases: first, a routing solution (the aggregation tree) which establishes the direction of transmissions, followed by the scheduling solution. The preferred scheduling process has been link activation model. We propose the inversion of these two phases and the adoption of the node activation model. In this new paradigm, it is not necessary to define first the logical topology and latter schedule transmissions, instead, schedule precedes the selection of the aggregation tree. This order inversion allows that interference restricts the directions (routing) that aggregated information can follow in order to reach its destination. We show that the use of node activation as a practical approach, being less computationally demanding than link activation. The result of a node schedule selection is a subset of suboptimal spanning trees, from whom it is possible select the optimal one. In contrast, the use of link activation is more demanding because, in general, a communication graph has more links than nodes.

✓ **Pipelined Aggregation Convergecast Problem**
We propose a variation of aggregation convergecast problem by relaxing the restriction of having to satisfy all precedence constraints within one single cycle. The use of a pipelined solution become possible by relaxing this restriction. This modification on the problem formulation allows aggregation convergecast solutions with higher throughput which are necessary for some classes of sensor applications. We define pipelining similarly to an assembly line composed of multiple serial stages. A final product must go through all stages. The construction of a new product can start on the initial stages as soon as its execution does not conflict with the assembly of the previous product, such that the assembly line can output a new product sooner than the system latency.

✓ **Algorithm for Pipelined Aggregation Convergecast using Reachability Constraints**
We propose an algorithm for pipelined aggregation convergecast. The algorithm does not require the selection of the logical topology in the first stage. Even though the logical topology is not initially select, the final solution preserves low node degree, a desirable topological property. We demonstrate, through numerical results, our proposed algorithm performance versus the state-of-the-art solution of aggregation con-

vergecast without pipeline, and also versus another pipelined aggregation convergecast where the logical topology is pre-selected. Even though the proposed algorithm is not providing the optimal throughput, our approach is able to present solutions with high throughput, at expense of some increase on delay.

6.2 Future Work

We present in Chapter 3 some evidence pointing out that SPT may not be the most suitable aggregation tree to achieve results close to the optimal. The presence of cliques in the communication graph also hinder SPT from creating a schedule with small number of time slots. We must add that most algorithmic solutions in the literature [24, 48, 76, 110, 113], after obtaining an aggregation tree, address the scheduling phase using a *bottom-up* approach, where the leaves in the logical topology are scheduled first, removed from the schedulable nodes, and a new set of nodes (taken from the new leaves created by the removal of previously scheduled leaves) are ready to be scheduled. This bottom-up approach privileges lower layer nodes of the graph, scheduling them first. The *two-phase approach* is extensively discussed in this Thesis. Routing and scheduling are not directly linked in a single heuristic or process.

The aggregation convergecast has a natural propensity to be viewed as a *all-to-one* and *bottom-up* problem. This view influences the heuristics proposed to address it. A different problem may have a different perspective. A different view is found in Minimal Delay Broadcast (MDB) [65, 82] problem. MDB has also a natural bias, however in the opposite direction: it presents a *top-down* and *one-to-all* pattern, which tends to influence its heuristics.

An evident work extension for our Thesis is to explore heuristics/algorithms for aggregation convergecast using a broadcast perspective. Such perspective includes a top-down algorithm and may combine routing and scheduling in a single heuristic. A top-down approach would create the aggregation tree and the schedule in a joint approach, relaxing the assumption previously held that SPT create close to optimal solutions. This work extension was investigated and we had the opportunity to co-participate in its execution. This work was developed by Matthias Jakob in his Master Thesis: *Time-efficient Scheduling for Aggregation Convergecast in Wireless Sensor Networks* [60]. We briefly present his main results and its consequences.

The idea of the Jakob's proposed Minimum Interference Network Topology (MINT) algorithm is to create an aggregation tree concurrently to selecting the transmission time, similarly to the way algorithms create broadcast trees. The algorithm starts with a single node as the current tree. This single node is the root of a growing tree. The natural choice is the sink node. The timeslot is set to one. Then, a candidate arc set is selected using arcs pointing to not yet connected nodes. The current timeslot is greedily filled with candidate arcs, if they do not interfere with previously scheduled ones. If further selection of arcs for the current timeslot is restricted by interference, the slot counter is increased and a new candidate arc set is obtained. The algorithm ends when there is no more arcs to be added. A simple reversion on the timeslot numbers is required to get the actual schedule assignment for aggregation convergecast.

We will only mention here the most successful strategy to select which candidate arcs will be first to be activated in the current timeslot. The most successful strategy is a dynamic one, where weights are attributed to each arc. The arc weight value is dependent not only on properties of the communication graph (such as node degree), but also on the current state of the algorithm construction. This dynamic and combined approach achieves the best results. Block count is a dynamic strategy which directly evaluates candidate arcs in relation to other candidates. This dynamic strategy is calculated as the amount of remaining arcs that would be blocked if the arc is added to the current timeslot. The arc with the lowest block count is added first, regardless if the arc has high or low source/destination degree, or child count. Figure 6.1 shows the comparison between several dynamic strategies. The low block count heuristic leads to the best results in comparison with other strategies. For comparison purposes, Figure 6.1 also has the results for WIRES algorithm.

The short schedule length results obtained using the low block count dynamic strategy produces a different tree shape than SPT: the outward fanning nodes, so characteristic to SPT solutions, do not appear with such consistency as when SPT is used. This fact results in a more homogeneous transmission distribution over the network. The low block count heuristic reduces the number of arcs used by each node (node degree), and cause a longer maximum path, as predicted by our results in Chapter 3, and also explored in Chapter 4. WIRES uses a form of SPT which balances node degree. However, the level of node degree balanced achieved by WIRES is inferior than the balance achieved in Jakob's work. The reason is that WIRES balances only nodes in the same layer, not nodes in the whole network. Figure 6.2 displays the node degree in the routing topologies of BSPT (used by

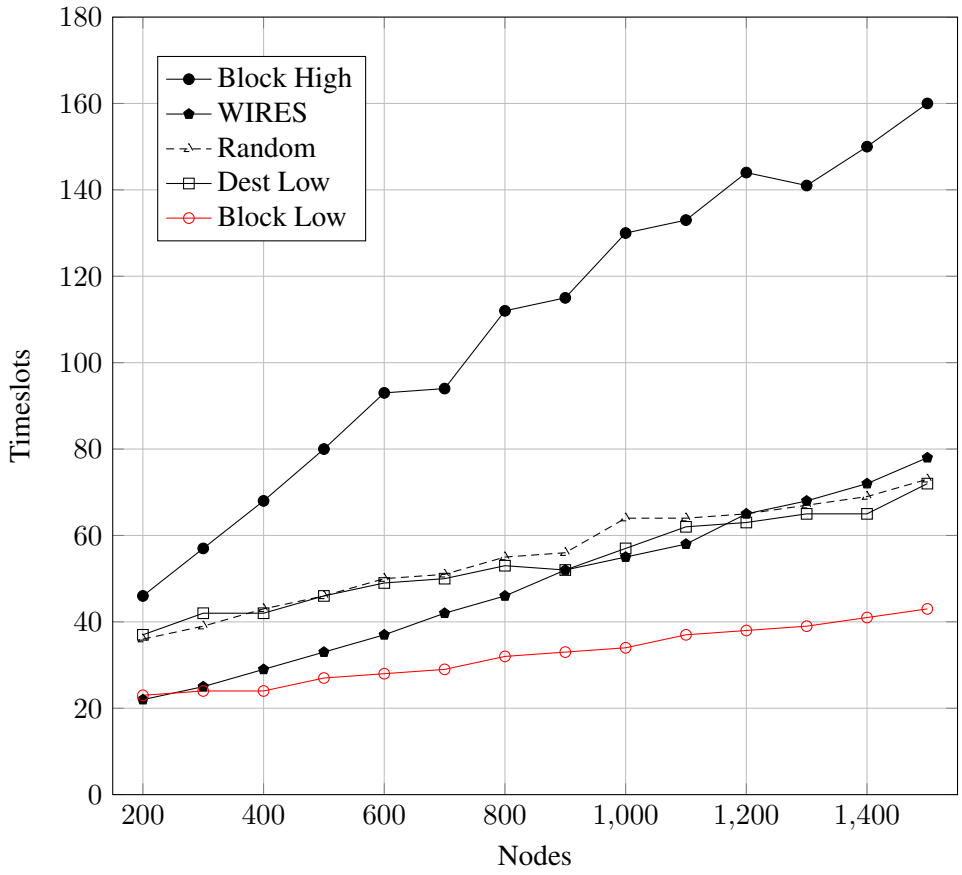
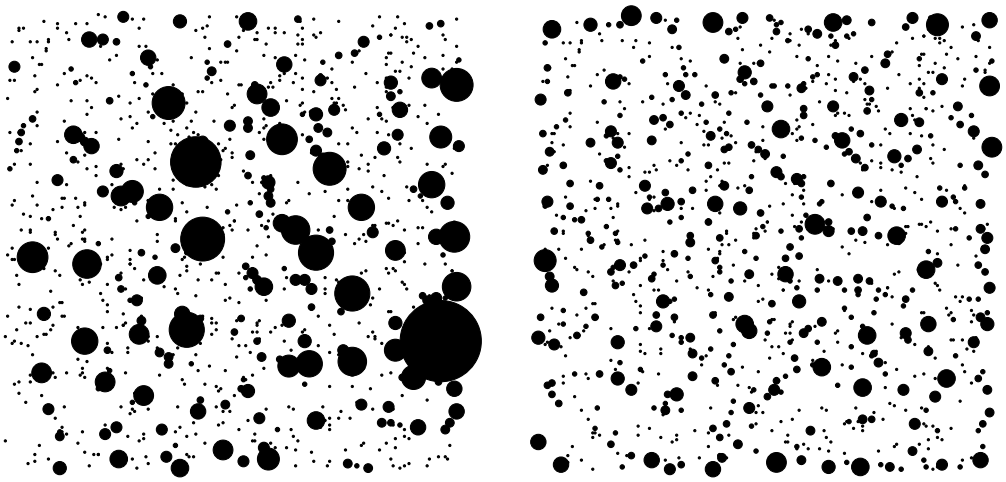


Figure 6.1: Schedule Length for Dynamic Strategies and WIRES
Taken from [60]

WIRES) and MINT (used by Jakob's algorithm). A larger point indicates a higher node degree.

This work extension confirms our observations about the limitation of using of SPT. Free from SPT limitation, this work explores a top-down approach to address the aggregation convergecast, similar to algorithms designed to minimum delay broadcast problem. The process resembles an outward arborescence process, which attempts to minimize network interference in every step of the topology construction. The schedule is committed as the tree is expanded. At the end of the tree construction, the schedule is reversed and the aggregation convergecast solution is ready. This aggregation convergecast solution breaks the insistence of two phase approach, where routing is addressed first.



(a) BSPT-WIRES (b) MINT

Figure 6.2: Node degree in the Routing Topologies
Taken from [60]

6.3 Future Directions

✓ **Real Setting Implementation**

This Thesis assumes a number of simplifying assumptions to reduce the complexity of selecting routing and scheduling for WSN applications. The new insights brought to light about routing and scheduling can be used to experiment on network simulators, or real settings. Real implementations avoid misleading or unrealistic conclusions [64].

✓ **Machine-Job Representation by Clique Decomposition**

It has been shown on Chapter 4 that an aggregation convergecast scheduling problem can be transformed into a machine-job representation. Multiprocessor Task Scheduling is the model that allows a direct representation of the restrictions of the aggregation convergecast scheduling problem. A machine in MTS represents the conflict for the execution of each task. This conflict is represented by the edges and arcs on the extended conflict graph. An immediate equivalence is to transform each edge and arc into an independent machine. The transformation can be executed by decomposing the extended conflict graph into cliques [71, 103], and naming each clique as a machine. The transformation is not explored in our Thesis and can be used as bridge between these two problems. Kramer *et al.*[16, 17, 66] have developed several algorithm for scheduling MTS problems, which we have the potential to use to address

aggregation convergecast problems.

✓ **Pipelinization of Single-Period Solutions**

There is a considerable amount of algorithms devoted to provide a solution for the aggregation convergecast problem restricted to the case where a complete snapshot collection must be executed in a single schedule period. It is not inconceivable to imagine that schedules under single-period restriction could be *pipelinezed*. The *pipelinization* would be the process of creating a pipelined schedule using as basis the aggregation tree and the not-pipelined schedule, instead of constructing a new solution from scratch. The possibility of obtaining significant results from the pipelinezation process is more likely in large networks, where the longest shortest path is likely to have a large amount of hops.

✓ **Pipelined Aggregation Convergecast Hardness**

Aggregation convergecast problem has a singular characteristic of combining two distinct aspects in a single solution: logical topology and scheduling. Chen *et al.*[25] proposed an elusive proof for the case of aggregation convergecast when the precedence constraints are confined to a single period. In [81], the authors provide a proof for aggregation convergecast hardness, however it accounts only for the scheduling part. It has being difficult to encompass in a single proof all aspects of aggregation convergecast. The difficulty lays in trying to format the proof as a network design problem [49], while it is also a sequencing and scheduling problem [49], or *vice versa*. For instance, graph coloring can be reduced to several scheduling problems. However, in the case of aggregation convergecast, defining the logical topology (or the connection between two colored nodes in graph coloring) is also part of the problem. Pipelined aggregation convergecast is a variation of the original aggregation convergecast problem, and it has the same difficult. A definitive hardness proof is still open to be provided.

✓ **Multiple-Radio Reception from MIMO Technologies**

The use of **MIMO** (Multiple Input, Multiple Output) Technologies [19] promisses performance enhancement. This technology may allow capacity gains over **SISO** (Single Input, Single Output) when *MIMO* is used in spatial multiplexing fashion. It could mean that single-radio nodes may cooperate on data transmission, or transceivers that could receive multiple transmissions concurrently. The possibility of multiple reception by the same node changes the model used on this Thesis, in particular in how

interference is understood and modeled. The consequence would be the possibility of schedules considering that a node may understand a transmission while receiving another transmission not intended for himself.

- ✓ **Multiple Phase Framework**

  A unified framework, composed of multiple phases: the request and the response of periodic data collection of sensed data, can be considered. One example of this framework is the dissemination of a *request* using some form of broadcast, and the *response*, in the form of a aggregation convergecast. A *combined* framework, coalescing request and response phases, can be examined to see if it can provide scheduling gains.

Bibliography

[1] Occurrence of the gasoline additive mtbe in shallow ground water in urban and agricultural areas. Technical Report 114, US Geological Survey, March 1995.

[2] C.J. Alpert, T.C. Hu, J.H. Huang, A.B. Kahng, and D. Karger. Prim-Dijkstra tradeoffs for improved performance-driven routing tree design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(7):890–896, July 1995.

[3] A. Amoura, E. Bampis, C. Kenyon, and Y. Manoussakis. Scheduling independent multiprocessor tasks. In Rainer Burkard and Gerhard Woeginger, editors, *Algorithms - ESA '97*, volume 1284 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 1997. 10.1007/3-540-63397-9.

[4] Min Kyung An, Nhat X. Lam, Dung T. Huynh, and Trac N. Nguyen. Minimum latency data aggregation in the physical interference model. *Computer Communications*, 35(18):2175 – 2186, 2012.

[5] George V. Andreev and Yuri N. Sotskov. A branch and bound method for mixed graph coloring and scheduling. In *Proceedings of the 16th International Conference on CAD/CAM Robotics & Factories of the Future (CARS & FOF 2000)*, volume 1, pages 1–8, Port of Spain / Trinidad and Tobago, June 2000.

[6] V. Annamalai, S.K.S. Gupta, and L. Schwiebert. On tree-based convergecasting in wireless sensor networks. In *Wireless Communications and Networking, WCNC 2003*, volume 3, pages 1942–1947, March 2003.

[7] Baruch Awerbuch, Alan Baratz, and David Peleg. Cost-sensitive analysis of communication protocols. In *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, PODC '90, pages 177–187, New York, NY, USA, 1990. ACM.

[8] D. J. Baker and Anthony Ephremides. The architectural organization of a mobile radio network via a distributed algorithm. *Communications, IEEE Transactions on*, 29(11):1694–1701, 1981.

[9] Amol Bakshi and Viktor K. Prasanna. Algorithm design and synthesis for wireless sensor networks. *Parallel Processing, International Conference on*, 0:423–430, 2004.

[10] Jorgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: Theory, Algorithms and Applications*, chapter Branchings, pages 339–372. Springer-Verlag London Limited, 2010.

[11] Kevin Barnhart, Inigo Urteaga, Qi Han, Anura Jayasumana, and Tissa Illangasekare. On integrating groundwater transport models with wireless sensor networks. *Ground Water*, 48(5):771–780, August 2010.

[12] Roman Bartk. Constraint-based scheduling: An introduction for newcomers. In *In Intelligent Manufacturing Systems 2003*, pages 69–74. IFAC Publications, Elsevier Science, 2003.

[13] Marc Benkert, Joachim Gudmundsson, Herman Haverkort, and Alexander Wolff. Constructing minimum-interference networks. *Computational Geometry*, 40(3):179 – 194, 2008.

[14] D.M. Blough, G. Resta, and P. Santi. Approximation algorithms for wireless link scheduling with SINR-based interference. *Networking, IEEE/ACM Transactions on*, 18(6):1701 –1712, December 2010.

[15] Peter Brucker. *Scheduling Algorithms*. Springer, 5th edition, 2007.

[16] Peter Brucker and Andreas Krmer. Shop scheduling problems with multiprocessor tasks on dedicated processors. *Annals of Operations Research*, 57:13–27, 1995. 10.1007/BF02099688.

[17] Peter Brucker and Andreas Krmer. Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems. *European Journal of Operational Research*, 90(2):214 – 226, 1996.

[18] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320, New York, NY, USA, October 2006. ACM.

[19] J. Burdin and J. Dunyak. Enhancing the performance of wireless sensor networks with mimo communications. In *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 2321–2326 Vol. 4, 2005.

[20] P. M. Camerini, L. Fratta, and F. Maffioli. A note on finding optimum branchings. *Networks*, 9(4):309–312, 1979.

[21] A. Capone, G. Carello, I. Filippini, S. Gualandi, and F. Malucelli. Routing, scheduling and channel assignment in wireless mesh networks: Optimization models and algorithms. *Ad Hoc Networks*, 8(6):545–563, August 2010.

[22] Antonio Capone, Giuliana Carello, Ilario Filippini, Stefano Gualandi, and Federico Malucelli. Solving a resource allocation problem in wireless mesh networks: A comparison between a CP-based and a classical column generation. *Networks*, 55(3):221–233, May 2010.

[23] Jianer Chen and Chung-Yee Lee. General multiprocessor task scheduling. *Naval Research Logistics (NRL)*, 46(1):57–74, 1999.

[24] Xujin Chen, Xiaodong Hu, and Jianming Zhu. Minimum data aggregation time problem in wireless sensor networks. In *Mobile Ad-hoc and Sensor Networks*, volume 3794 of *Lecture Notes in Computer Science*, pages 133–142. Springer Berlin / Heidelberg, 2005.

[25] Xujin Chen, Xiaodong Hu, and Jianming Zhu. Data gathering schedule for minimal aggregation time in wireless sensor networks. *Int. J. Distrib. Sen. Netw.*, 5(4):321–337, July 2009.

[26] Zhengyu Chen, Geng Yang, Lei chen, and Jin Wang. An algorithm for data aggregation scheduling with long-lifetime and low-latency in wireless sensor networks. *International Journal of Future Generation Communication and Networking*, 5(4), December 2012.

[27] J. Cheriyan and R. Ravi. Approximation algorithms for network problems. http://www.gsia.cmu.edu/andrew/ravi, September 1998. Lecture Notes.

[28] I. Chlamtac, A. Farago, and Hye Yeon Ahn. A topology transparent link activation protocol for mobile CDMA radio networks. *Selected Areas in Communications, IEEE Journal on*, 12(8):1426–1433, October 1994.

[29] CY Chong and SP Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, August 2003.

[30] Y.J. Chu and T.H. Liu. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400, 1965.

[31] Jens Clausen. Branch and bound algorithms - principles and examples. Department of Computer Science, University of Copenhagen, March 1999.

[32] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[33] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 340 Pine Street, San Francisco, 2003.

[34] Reinhard Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, July 2010.

[35] M. Ding, X. Cheng, and G. Xue. Aggregation tree construction in sensor networks. In *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, volume 4, pages 2168–2172, October 2003.

[36] Maciej Drozdowski. Scheduling multiprocessor tasks - an overview. *European Journal of Operational Research*, 94(2):215 – 230, 1996.

[37] Hongwei Du, Zhao Zhang, Weili Wu, Lidong Wu, and Kai Xing. Constant-approximation for optimal data aggregation with physical interference. *Journal of Global Optimization*, pages 1–14, 2012. 10.1007/s10898-012-9939-7.

[38] O. Durmaz Incel, A. Ghosh, B. Krishnamachari, and K. Chintalapudi. Fast data collection in tree-based wireless sensor networks. *Mobile Computing, IEEE Transactions on*, PP(99):1, February 2011.

[39] Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

[40] A. Ephremides and T.V. Truong. Scheduling broadcasts in multihop radio networks. *Communications, IEEE Transactions on*, 38(4):456–460, April 1990.

[41] Sinem Ergen and Pravin Varaiya. Tdma scheduling algorithms for sensor networks. Technical report, University of California, Berkeley, July 2005.

[42] Deborah Estrin. Embedded networked sensing for environmental monitoring: Applications and challenges. Sensor Networks Seminar, 2004.

[43] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-network aggregation techniques for wireless sensor networks: a survey. *Wireless Communications, IEEE*, 14(2):70–87, April 2007.

[44] Gustavo B. Figueiredo, Nelson L. S. da Fonseca, and José A. S. Monteiro. A minimum interference routing algorithm with reduced computational complexity. *Comput. Netw.*, 50(11):1710–1732, August 2006.

[45] Martin Fussen, Roger Wattenhofer, and Aaron Zollinger. On interference reduction in sensor networks. Technical reports 453, ETH, Department of Computer Science, 2004.

[46] Harold Gabow, Zvi Galil, Thomas Spencer, and Robert Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:109–122, 1986.

[47] A.El. Gamal, J. Mammen, B. Prabhakar, and D. Shah. Throughput-delay tradeoff in wireless networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 4 vol. (xxxv+2866), Mar 2004.

[48] Shashidhar Gandham, Ying Zhang, and Qingfeng Huang. Distributed time-optimal scheduling for convergecast in wireless sensor networks. *Computer Networks*, 52(3):610 – 629, 2008.

[49] Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, January 1979.

[50] Leonidas Georgiadis. Arborescence optimization problems solvable by Edmonds' algorithm. *Theoretical Computer Science*, 71:233–240, May 2003.

[51] A. Ghosh, O.D. Incel, V.S.A. Kumar, and B. Krishnamachari. Bounded-degree minimum-radius spanning trees for fast data collection in sensor networks. In *INFOCOM IEEE Conference on Computer Communications Workshops , 2010*, pages 1 –2, march 2010.

[52] A. Ghosh, O.D. Incel, V.S.A. Kumar, and B. Krishnamachari. Multichannel scheduling and spanning trees: Throughput;delay tradeoff for fast data collection in sensor networks. *Networking, IEEE/ACM Transactions on*, 19(6):1731–1744, Dec 2011.

[53] V. Gordon, M. Kovalyov, G. Levin, Y. Shafransky, Y. Sotskov, V. Strusevich, and A. Tuzikov. Vyacheslav tanaev: contributions to scheduling and related areas. *Journal of Scheduling*, pages 1–16, Mar 2011. 10.1007/s10951-011-0230-4.

[54] Jimmi Grönkvist. Assignment methods for spatial reuse tdma. In *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '00, pages 119–124, Piscataway, NJ, USA, 2000. IEEE Press.

[55] Yongpei Guan, Wen-Qiang Xiao, Raymond K Cheung, and Chung-Lun Li. A multiprocessor task scheduling model for berth allocation: heuristic and worst-case analysis. *Operations Research Letters*, 30(5):343 – 350, 2002.

[56] B. Hajek and G. Sasaki. Link scheduling in polynomial time. *Information Theory, IEEE Transactions on*, 34(5):910 –917, sep 1988.

[57] Pierre Hansen, Julio Kuplinsky, and Dominique de Werra. Mixed graph colorings. *Mathematical Methods of Operations Research*, 45:145–160, 1997. 10.1007/BF01194253.

[58] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. pages 607–613. Morgan Kaufmann, 1995.

[59] Ozlem Incel, Amitabha Ghosh, and Bhaskar Krishnamachari. Scheduling algorithms for tree-based data collection in wireless sensor networks. In Sotiris Nikoletseas and Jos D.P. Rolim, editors, *Theoretical Aspects of Distributed Computing in Sensor Networks*, number 4 in Monographs in Theoretical Computer Science, pages 407–445. Springer Berlin / Heidelberg, 2011.

[60] Matthias Jakob. Time-efficient scheduling for aggregation convergecast in wireless sensor network. Master's thesis, Ludwig Maximilians Universitat Munchen, Aug 2012.

[61] R. M. Karp. A simple derivation of Edmonds' algorithm for optimum branchings. *Networks*, 1(3):265–272, 1971.

[62] S. Khuller, B. Raghavachari, and N. Young. Balancing minimum spanning trees and shortest-path trees. *Algorithmica*, 14:305–321, 1995.

[63] S. Kompella, J.E. Wieselthier, A. Ephremides, H.D. Sherali, and G.D. Nguyen. On optimal SINR-based scheduling in multihop wireless networks. *Networking, IEEE/ACM Transactions on*, 18(6):1713 –1724, December 2010.

[64] David Kotz, Calvin Newport, Robert S. Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental evaluation of wireless simulation assumptions. Technical Report TR2004-507, Dartmouth College, Computer Science, Hanover, NH, June 2004.

[65] Dariusz Kowalski and Andrzej Pelc. Optimal deterministic broadcasting in known topology radio networks. *Distributed Computing*, 19:185–195, 2007. 10.1007/s00446-006-0007-8.

[66] Andreas Kramer. *Scheduling Multiprocessor Tasks on Dedicated Processors*. PhD thesis, Fachbereich Mathematik/Informatik - Universitat Osnabruck, Feb 1995.

[67] S. Kwon and N. B. Shroff. Energy-efficient interference-based routing for multi-hop wireless networks. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1 –12, april 2006.

[68] Jochen Knemann, Asaf Levin, and Amitabh Sinha. Approximating the degree-bounded minimum diameter spanning tree problem. *Algorithmica*, 41:117–129, 2005. 10.1007/s00453-004-1121-2.

[69] Philippe Laborie. Ibm ilog cp optimizer for detailed scheduling illustrated on three problems. In *Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, CPAIOR '09, pages 148–162, Berlin, Heidelberg, 2009. Springer-Verlag.

[70] Chung-Yee Lee, Lei Lei, and Michael Pinedo. Current trends in deterministic scheduling. *Annals of Operations Research*, 70:1–41, 1997. 10.1023/A:1018909801944.

[71] Hanns-Georg Leimer. Optimal decomposition by clique separators. *Discrete Mathematics*, 113(1-3):99 – 123, 1993.

[72] Deying Li, Qinghua Zhu, Hongwei Du, and Jianzhong Li. An improved distributed data aggregation scheduling in wireless sensor networks. *Journal of Combinatorial Optimization*, pages 1–20, 2012. 10.1007/s10878-012-9504-9.

[73] Xiang-Yang Li, XiaoHua Xu, ShiGuang Wang, ShaoJie Tang, GuoJun Dai, JiZhong Zhao, and Yong Qi. Efficient data aggregation in multi-hop wireless sensor networks under physical interference model. In *Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on*, pages 353 –362, Oct 2009.

[74] C. Liu, X. Li, S. Muthukumar, H. Gill, T. Saeed, B. Loo, and Prithwish Basu. A policy-based constraint-solving platform towards extensible wireless channel selection and routing. In *ACM Workshop on Programmable Routers for Extensible Services of TOmorrow (PRESTO)*, December 2010.

[75] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Operating Systems Review*, 36:131–146, December 2002.

[76] Baljeet Malhotra, Ioanis Nikolaidis, and Mario Nascimento. Aggregation convergecast scheduling in wireless sensor networks. *Wireless Networks*, 17:319–335, Feb 2011.

[77] T. Moscibroda and R. Wattenhofer. The complexity of connectivity in wireless networks. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1 –13, April 2006.

[78] Thomas Moscibroda. The worst-case capacity of wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 1–10, New York, NY, USA, 2007. ACM.

[79] M.J. Neely and E. Modiano. Capacity and delay tradeoffs for ad hoc mobile networks. *Information Theory, IEEE Transactions on*, 51(6):1917 – 1937, Jun 2005.

[80] R.J. Nemzek, J.S. Dreicer, and D.C. Torney. Distributed sensor networks for detection of mobile radioactive sources. In *Nuclear Science Symposium Conference Record, 2003 IEEE*, volume 3, pages 1463–1467, October 2003.

[81] Meng-Shiuan Pan and Yu-Chee Tseng. Quick convergecast in zigbee beacon-enabled tree-based wireless sensor networks. *Computer Communications*, 31(5):999 – 1011, 2008. Mobility Management and Wireless Access.

[82] David Peleg and Tomasz Radzik. Time-efficient broadcast in radio networks. In Arie Koster and Xavier Muoz, editors, *Graphs and Algorithms in Communication Networks*, Texts in Theoretical Computer Science. An EATCS Series, pages 311–334. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-02250-0-12.

[83] Michael Pinedo. *Scheduling: Theory, Algorithms and Systems*. Springer, fourth edition, Jan 2012.

[84] S. Ramanathan and E.L. Lloyd. Scheduling algorithms for multihop radio networks. *Networking, IEEE/ACM Transactions on*, 1(2):166 –177, April 1993.

[85] R. Ramaswami and K.K. Parhi. Distributed scheduling of broadcasts in a radio network. In *INFOCOM '89. Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies. Technology: Emerging or Converging, IEEE*, pages 497 –504 vol.2, april 1989.

[86] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt, III. Many birds with one stone: multi-objective approximation algorithms. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 438–447, New York, NY, USA, 1993. ACM.

[87] Bernard Ries. Coloring some classes of mixed graphs. *Discrete Applied Mathematics*, 155(1):1 – 6, 2007.

[88] Bernard Ries and D. de Werra. On two coloring problems in mixed graphs. *European Journal of Combinatorics*, 29(3):712 – 725, 2008.

[89] Francesca Rossi, Peter van Beek, and Toby Walsh. Introduction to constraint programming. In *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[90] Nathan Carl Rowe. Distributed radiation monitoring via a secure wireless sensor platform. Master's thesis, University of Tennessee, Knoxville, December 2008.

[91] R. Sappidi, A. Girard, and C. Rosenberg. Maximum achievable throughput in a wireless sensor network using in-network computation for statistical functions. *Networking, IEEE/ACM Transactions on*, PP(99):1, 2012.

[92] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. *Modeling and Programming with Gecode*. KTH - Royal Intitute of Technology, Sweden, 3.4.2 edition, October 2010.

[93] Weiping Shang, Pengjun Wan, and Xiaodong Hu. Approximation algorithm for minimal convergecast time problem in wireless sensor networks. *Wireless Networks*, 16:1345–1353, 2010. 10.1007/s11276-009-0207-9.

[94] Yi Shi, Y. Thomas Hou, Jia Liu, and Sastry Kompella. How to correctly use the protocol interference model for multi-hop wireless networks. In *Proceedings of the 10th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '09, pages 239–248, New York, NY, USA, 2009. ACM.

[95] Helmut Simonis. Challenges for constraint programming in networking, 2005.

[96] Steven S. Skiena. *The Algorithm Design Manual*. Springer-Verlag New York, Inc., 1998.

[97] Yu.N Sotskov. Mixed multigraph approach to scheduling jobs on machines of different types. *Optimization*, 42(3):245–280, 1997.

[98] Evandro Souza and Ioanis Nikolaidis. An exploration of aggregation convergecast scheduling. *Ad Hoc Networks*. Accepted for Publication.

[99] Evandro Souza and Ioanis Nikolaidis. Modeling aggregation convergecast scheduling using constraints. In *Proceedings of the 14th ACM international conference on*

Modeling, analysis and simulation of wireless and mobile systems, MSWiM '11, pages 231–234, New York, NY, USA, 2011. ACM.

[100] Evandro Souza and Ioanis Nikolaidis. On the application of pipelining in aggregation convergecast scheduling. In *14th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (IEEE WoWMoM 2013)*, Madrid, Spain, June 2013.

[101] V. S. Tanaev, Y. N. Sotskov, and V. A. Strusevich. *Scheduling Theory. Multi-Stage Systems*, volume 285 of *Mathematics and Its Applications*. Kluwer Academic Publishers, P.O. Box 17,3300 AA Dordrecht, The Netherlands., 1994.

[102] R. E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

[103] Robert E. Tarjan. Decomposition by clique separators. *Discrete Mathematics*, 55(2):221 – 232, 1985.

[104] Ali Tofigh. Optimum branchings and spanning aborescences. http://www.cvut.cz/, September 2009. Advanced Algorithms course notes, Available at Czech Technical University in Prague, http://www.cvut.cz.

[105] S. Toumpis and A.J. Goldsmith. Large wireless networks under fading, mobility, and delay constraints. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 4 vol. (xxxv+2866), Mar 2004.

[106] S. Upadhyayula and S.K.S. Gupta. Spanning tree based algorithms for low latency and energy efficient data aggregation enhanced convergecast (dac) in wireless sensor networks. *Ad Hoc Networks*, 5(5):626 – 648, 2007.

[107] Markus Volker. *Scheduling and Topology Control in Wireless Sensor Networks*. PhD thesis, Institut fur Theoretische Informatik, Universitat Karlsruhe, October 2008.

[108] P. von Rickenbach, S. Schmid, R. Wattenhofer, and A. Zollinger. A robust interference model for wireless ad-hoc networks. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, page 8 pp., april 2005.

[109] Dorothea Wagner and Roger Wattenhofer. *Algorithms for Sensor and Ad Hoc Networks*, volume 4621 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007.

[110] Peng-Jun Wan, Scott C.-H. Huang, Lixin Wang, Zhiyuan Wan, and Xiaohua Jia. Minimum-latency aggregation scheduling in multihop wireless networks. In *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '09, pages 185–194, New York, NY, USA, 2009. ACM.

[111] Guanyu Wang, Qiang-Sheng Hua, and Yuexuan Wang. Minimum latency aggregation scheduling for arbitrary tree topologies under the sinr model. In *Ad-hoc, Mobile, and Wireless Networks*, volume 7363 of *Lecture Notes in Computer Science*, pages 139–152. Springer Berlin / Heidelberg, 2012.

[112] B. Yu, J. Li, and Y. Li. Distributed data aggregation scheduling in wireless sensor networks. In *INFOCOM 2009, IEEE*, pages 2159 –2167, Apr 2009.

[113] Xingbo Yu, Sharad Mehrotra, and Nalini Venkatasubramanian. Sensor scheduling for aggregate monitoring in wireless sensor networks. In *Proceedings of the 19th International Conference on Scientific and Statistical Database Management*, SSDBM '07, page 24, Washington, DC, USA, July 2007.

[114] Jianming Zhu and Xiaodong HU. Improved algorithm for minimum data aggregation time problem in wireless sensor networks. *Journal of Systems Science and Complexity*, 21:626–636, 2008. 10.1007/s11424-008-9139-1.

[115] Marco Zimmerling. *Automatic Parameter Optimization of Sensor Network MAC Protocols*. PhD thesis, Faculty of Computer Science, Technische Universitat Dresden, Uppsala, Sweeden, August 2009.

Appendix A

Graph Theory Concepts

This appendix reviews the notation, definitions and elementary algorithms of Graph Theory, as they are applied in the context of this Thesis. The concepts follow the standard understanding in the literature [34, 10].

Definition A.1. ***(Undirected) Graph***
A graph is defined as a tuple $G = (V, E)$ consisting of a finite set V of vertices (or nodes), and a finite edge set E. An edge is a tuple $e = [u, v]$ with u and v being distinct elements of V . The edge set E defines a logic relation on V , and edges represent connections (or links) between vertices.

Definition A.2. ***Directed Graph***
A directed graph (or just digraph) D consists of a non-empty finite set V of elements called vertices and a finite set A of ordered pairs of distinct vertices called arcs. We call V the vertex set and A the arc set of D. We will often write $D = (V, A)$ which means that V and A are the vertex set and arc set of D, respectively. The ***order*** *(size) of D is the number of vertices (arcs) in D; the order of D will be sometimes denoted by $|D|$. The notations (u, v) represent an arc from vertex u directed to vertex v.*

Definition A.3. ***Simple Graph***
Simple graphs are graphs or digraphs not containing self loops or multiple edges. A self loop is an edge originating from and ending in the same vertex. Multiple edges join the same two vertices. A graph containing these elements is referred to as a multi graph or pseudo graph. If not stated otherwise, graphs throughout this thesis are considered to be simple graphs

Definition A.4. ***Planar Graph***
An undirected graph $G = (V, E)$ is planar if there exists a mapping f which maps G to $\Re^2$ in the following way: each vertex is mapped to a point in $\Re^2$ and distinct vertices are mapped to distinct; and each edge $(u, v) \in E$ is mapped to a simple (that is, not self-intersecting) curve C_{uv} from $f(u)$ to $f(v)$ and no two curves corresponding to distinct edges intersect, except possibly at their endpoints.

Definition A.5. ***Unit Disk Graph***
A unit disk graph is the intersection graph of a family of unit disks in the Euclidean plane. In a unit disk graph, there is an edge between two vertices u and v if and only if the Euclidean distance between u and v is at most 1. Equivalently, each vertice is identified with a disk of unit radius $r = 1$ in the plane, and is connected to all nodes within (or on the edge of) its corresponding disk.

Definition A.6. ***Path***
*A path is a sequence of distinct vertices $P = \{v_0, \cdots, v_k\}$ of a graph $G = (V, E)$ with edges $[v_i, v_{i+1}] \in E \quad for \quad 0 \leq i < k$. When there is a path P between two vertices they are referred to as connected by this path in G. The **length** of a path is the number of edges of the path. The path between u and v with the lowest possible length is called the shortest path from u to v. If $P = \{v_0, \cdots, v_{k-1}\}$ is a path and $k \leq 3$, the the $P' = P + [v_{k-1}, v_0]$ is called a **cycle**. A **directed path** is a path of a directed graph, where a sequence of arcs connects a sequence of vertices, always is the same direction.*

Definition A.7. ***Connected Graph***
A connected graph is a graph $G = (V, E)$ where each pair of vertices of V is connected by a path in G.

Definition A.8. ***Degree***
*The degree of a vertex u of a graph G is the number of edges incident to the vertex u. In a directed graph D, the number of arcs in A, whose tail vertex is u is called **out-degree**, while the number of arcs in A whose head vertex is u is known as **in-degree**.*

Definition A.9. ***Clique***
*In an undirected graph, **clique** is a subset of its vertices such that every two vertices in the subset are connected by an edge. A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex, that is, a clique which does not exist exclusively within the vertex set of a larger clique. A **maximum clique** is a clique of the largest possible size in a given graph. The **clique number** of a graph G is the number of vertices in a maximum clique in G.*

Definition A.10. ***Reachability***
*Reachability is the ability to get from one vertex in a directed graph to some other vertex. For a directed graph $D = (V, A)$, the reachability relation of D is the transitive closure of its arc set A. **Transitive closure** is the set of all ordered pairs (u, v) of vertices in V for which there exist a directed path from vertex u to vertex v.*

Definition A.11. ***Tree***
*A tree is an undirected connected graph $T = (V, E)$ without cycles. A tree can have a special vertex, called the **root** of the tree, and is then called rooted tree $T = (V, E, s)$.*

Definition A.12. ***Spanning Tree***
*A spanning tree of a graph $G = (V, E)$ is a subgraph $T = (V, E')$ of G where E' is a subset of $E(G)$. Every spanning tree of G has exactly $|V| - 1$ edges and for each vertex there is a unique path to the root vertice. A subgraph T_D of a connected directed graph D is a spanning oriented tree of D if the underlying graph T_D is a spanning tree in underlying graph D. A subdigraph T_D of a digraph D is an **in-branching** or **out-branching** if T_D is a spanning oriented tree of D and T_D has only one vertex s of out-degree (in-degree) zero. The vertex s is the **root** of T_D .*

Definition A.13. ***Arborescence***
*An arborescence is a cycle free directed graph where all vertices can be reached by a single path from a root vertice (or all vertices can reach the root vertex). An **out-arborescence** rooted at s is an oriented tree T_D which is not necessarily spanning such that $s \in V(T)$ and every vertex $u \in \{V(T) - s\}$ has in-degree 1. An **in-arborescence** with root s is defined analogously.*

Definition A.14. ***Shortest Path Tree***
*A spanning tree connecting each vertice u from a connected, undirected graph $G = (V, E)$ with a root s by the shortest possible path is called a shortest path tree (SPT). Analogously, a **Non-SPT** is a spanning tree connecting each vertice u to the root s where at least one path $P(u_i, \cdots, s)$ does not have minimal length.*

Definition A.15. ***Breadth First Search***
Given a graph $G = (V, E)$ and a distinguished root vertex s, breadth-first search systematically explores the edges of G to discover every vertex u that is reachable from s. Search starts at a root vertex s and continues breadth wise, which means that vertices are visited with increasing level. The union of edges used for the breadth first search and the vertex set of the graph result in a spanning tree of the graph where vertices are connected to the root by a shortest path.