Complexity in Constraint Satisfaction and Automated Configuration

Konstantinos Koiliaris

Oriel College

University of Oxford

A thesis submitted in partial fulfillment of the MSc in

Computer Science

September 2, 2011

This work is dedicated to my family
for all the sacrifices they made
for me to study at Oxford, the
city of Dreaming Spires.

Acknowledgements

I would like to thank Dr. Conrad Drescher for his excellent support and encouraging co-supervision throughout this dissertation. I would also like to thank Professor Georg Gottlob for giving me this wonderful subject and for giving me inspiration and motivation in times of distress.

Abstract

The Partner Units Problem (Pup) is a new benchmark configuration problem. This problem involves the configuration of a network of sensors and controllers, and has drawn significant attention due to the amount of industrial applications it finds. In this dissertation, we further explored previous work done on a tractable class of the problem, exploiting the notion of a path decomposition, representing and re-evaluating the encodings for the general version of the problem. During this endeavor, through constraint satisfaction methods, we presented new implied constraints and search conditions, which resulted in a number of results that give us new insight into the problem. Next, we extensively presented all the classes of the problem and analyzed their complexity, a problem that had been left open. Interestingly enough, the discrepancy between the classes of the problem was significant in terms of their structural properties. The complexity analysis showed that all the non trivial classes seem to belong in NP-Complete (some were proven and others were conjectured), and even the most trivial classes of the problem were proven to be solvable only in PTime. Finally, we presented a logical approach to the Pup; we used two algorithmic meta-theorems to approach and tackle the complexity of our unsolved classes. In doing so we discovered a new configuration problem, the Partner Units Embedding Problem, which we analyzed and proved to be fixed-point linear in respect to the treewidth of the input graphs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automated configuration can, informally, be defined as "special case of design activity, where the artifact being configured is assembled from instances of a fixed set of well-defined component types which can be composed conforming to a set of constraints"[32]. In a more abstract way, we conceive it as the "problem of building something from predefined parts with certain properties". One of the most popular approaches to tackling problems in automated configuration is through Constraint Satisfaction Problems (Csp) modeling.

Constraint Satisfaction Problems are problems that involve a set of variables, their domains and a set of constraints. We try to give our variables proper values from their domain such that no constraint is being violated, this area is known as constraint satisfaction. Typical example of CSP is the boolean satisfiability problem (SAT), where the question is whether the variables of a given logical formula φ can be assigned "true" or "false" values in such a way as to for the whole formula to evaluate to true.

This work is dedicated to a new configuration problem that has vast applications in the aforementioned areas and it is a combination of theory and practice; the Partner Units Problem (Pup). The Pup is not only an interesting theoretical problem; it finds numerous applications in the areas of surveillance, monitoring and security. Basically, any scenario that would involve communication between zones and attached sensors can be efficiently encoded as a Pup instance.

In the coming chapters, we will, on the theory part, analyze the nature of the problem by describing previous work done on it and giving new results to promote intuition and understanding. We clearly define the frontier line of the research of this problem and then go on to expand it. In this endeavor, we mainly go through the fields of configuration, graph theory and finally logic. We analyze the structure of the input graphs to see how they affect the tractability and computability of the problem. Moreover, we classify the problem to all of its subproblems based on one of the parameters it has. Next, we try to tackle the complexity of the problem in all of its versions — a challenging task indeed. It appears that the most of the versions of the problem are hard with only its most trivial cases being polynomial. Finally, we present a logical approach to the problem. We show that the problem cannot be encoded in Monadic Second Order Logic (MSOL). But, we prove that a very close variation of the problem can be expressed in MSOL and hence is a fixed-parameter tractable.

On the more practical side of the project, we attempt to improve the runtime of the existing encodings and algorithm. We try to achieve that by looking at the algorithm / encodings themselves, trying to add or modify the constraints they use. The main idea here is to have the algorithm tap as many constraints as possible early on, in order to reduce the choices it will have to make afterwards. So, we added more constraints, to try and prune the search space of the encodings. We believe that those new conditions and constraints if applied to the encodings at hand will improve their efficieency.

1.1 Aim

The aim of this dissertation is to further advance the state-of-the-art of the newly introduced Partner Units Problem. We aim at presenting previous work done and then expanding it, both in an applied scope, by adding new constraints for the encodings of the problem, and in a theoretical perspective where we will be proving the complexity of several classes of this configuration problem. Finally, the work from this dissertation will look at proving a logical approach to the Pup and by exploiting two major algorithmic meta-theorems will aspire to classify the remaining versions.

1.2 The Partner Units Problem

We will at this point quote the introduction from [3]:

"The Partner Units Problem has recently been proposed as a new benchmark configuration problem [20]. It captures the essence of a specific type of configuration problem that frequently occurs in industry.

Informally the Pup can be described as follows: consider a set of sensors that are grouped into zones. A zone may contain many sensors, and a sensor may be attached to more than one zone. The Pup then consists of connecting the sensors and zones to control units, where each control unit can be connected to the same fixed maximum number *UnitCap* of zones and sensors.[1] Moreover, if a sensor is attached to a zone, but the sensor and the zone are assigned to different control units, then the two control units in question have to be (directly) connected. However, a control unit cannot be connected to more than *InterUnitCap* other control units (the partner units).

For an application scenario consider e.g. a museum where we want to keep track of the number of visitors that populate certain parts (zones) of the building. To this end, the doors leading from one zone to another are equipped with sensors. To keep track of the visitors, the zones and sensors are attached to control units; the adjacency constraints on the control units ensure that communication between control units can be kept simple.

It is worth emphasizing that the Pup is not limited to this application domain: It occurs whenever sensors that are grouped into zones have to be attached to control units, and communication between units should be kept simple - e.g. in intelligent traffic management, or surveillance and security applications. The Pup is used as a novel benchmark instance at the 2011 answer set programming competition [5].

Figure 1.1 shows a Pup instance and a solution for the special case where $\mathit{UnitCap} = \mathit{InterUnitCap} = 2$ — six sensors (left) and six zones (right) which are completely inter-connected are partitioned into units (shown as squares) respecting the adjacency constraints. Note that for the given parameters this is a maximal solvable instance; it is not possible to connect a new zone or sensor to any of the existing ones.

In this dissertation we will show that the case where $\mathit{InterUnitCap} = 2$ and $\mathit{UnitCap} = k$ for some fixed k is tractable by giving a specialized NLogSpace algorithm that is based on the notion of a path decomposition. While already this case is of great importance for our partners in industry, the general case is quite interesting in its own right: Consider, for instance, a grid of rooms, where every room is accessible from each neighboring room, and all the doors are fitted with a sensor. Moreover, there are doors to the outside on two sides of the building; the respective instance is shown in figure 1.2 with rooms as black squares, and doors as cyan circles. It is not hard to see that this instance is unsolvable for $\mathit{InterUnitCap} = 2$ and $\mathit{UnitCap} = 2$, whereas it is easily solved for $\mathit{InterUnitCap} = 4$ and $\mathit{UnitCap} = 2$: Every room goes on a distinct unit, together with the sensors to the west and to the north; the connections between units correspond to those between rooms. Clearly this solution is optimal in the sense of using the least possible number of units.

[1]For ease of presentation and without loss of generality we assume that *UnitCap* is the same for zones and sensors.

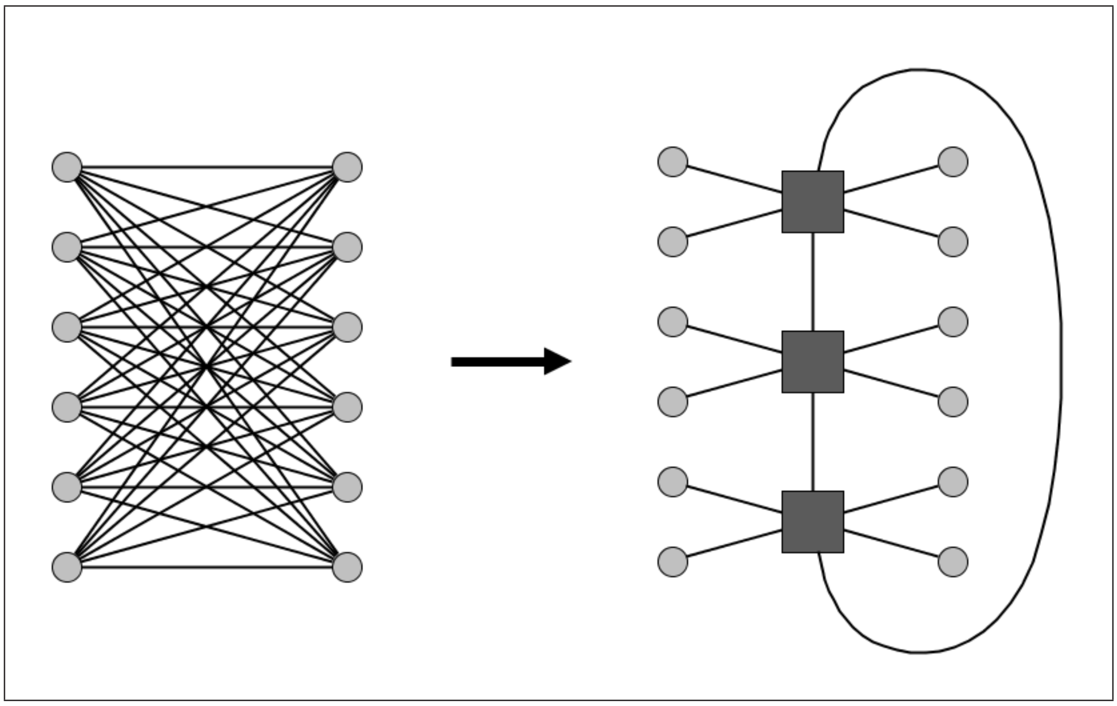

Figure 1.1: Solving a $K_{6,6}$ Partner Units Instance — Partitioning Sensors and Zones into Units on a Circular Unit Layout

Hence, in this work, we will also present encodings in the optimization frameworks of constraint satisfaction programming that can be used to solve the general version of the PUP. We also show how to adapt these encodings to the special case $\mathit{InterUnitCap} = 2$. For both cases we evaluate the performance.

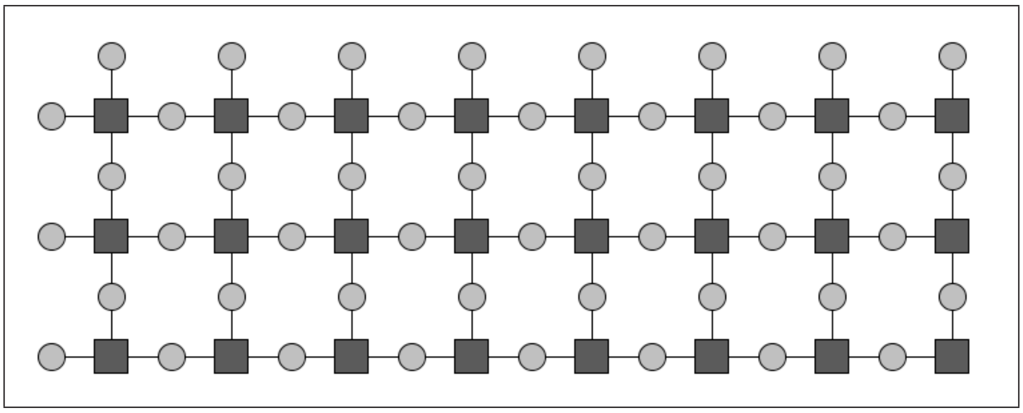

Figure 1.2: A Grid-like PUP Instance

1.3 Formal Definition of the Pup

In this section we present the basic formal definitions of the Pup, some basic properties of Pup instances that affect the existence, shape, or size of solutions, as well as basic complexity results.

Formally, the Pup consists of partitioning the vertices of a bipartite graph $G = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$ into a set $\mathcal{U}$ of bags such that each bag

- contains at most *UnitCap* vertices from $\mathcal{V}_1$ and at most *UnitCap* vertices from $\mathcal{V}_2$; and
- has at most *InterUnitCap* adjacent bags, where the bags U_1 and U_2 are adjacent whenever $V_i \in U_1$ and $V_j \in U_2$ and $(V_i, V_j) \in \mathcal{E}$.

To every solution of the Pup we can associate a solution graph. For this we associate to every bag $U_i \in \mathcal{U}$ a vertex V_{U_i}. Then the solution graph G^* has the vertex set $\mathcal{V}_1 \cup \mathcal{V}_2 \cup \{V_{U_i} \mid U_i \in \mathcal{U}\}$ and the set of edges $\{(V, V_{U_i}) \mid V \in U_i \wedge U_i \in \mathcal{U}\} \cup \{(V_{U_i}, V_{U_j}) \mid U_i \text{ and } U_j \text{ are adjacent.}\}$. In the following we will refer to the subgraph of the solution graph induced by the v_{U_i} as the *unit graph*.

The following are the two most important reasoning tasks for the Pup: decide whether there is a solution, and find an optimal solution, that is, one that uses the minimal number of control units. We are especially interested in the latter problem. For this we consider the corresponding decision problem: Is there a solution with a specified number of units? The rationale behind the optimization criterion is that (a) units are expensive, and (b) connections are cheap."

1.4 Thesis Structure

The remainder of this dissertation is organized as follows:

- In Chapter 2 we present the research frontier on the Pup underlining all the crucial previous results, based on the work done in [3].
- In Chapter 3 we introduce the results that aim at improving previous algorithms and encodings.
- In Chapter 4 we classify the different versions of the Pup and analyze their complexity, proving it for some of the cases.
- In Chapter 5 we give a logic approach to the Pup and present results around Courcelle's and Arnborg et al.'s Meta Theorems.

- In Chapter 6 we conclude this work by underlining the impact of the work presented, noting the future work and pinpointing the open problems.

Chapter 2

Previous Results and Basic Facts

2.1 The General Case of the Pup

2.1.1 The Pup is Hard

In this section we will provide basic facts and previous results on the general version of the Pup. Let us start by saying that the Partner Units Problem is in fact hard. After having worked on the problem for several months, the author would like to state the following conjecture on the problem's tractability,

Conjecture 2.1.1. *The* Pup *in its general cases is* NP-Complete*.*

Next, let us look into the solutions of the Pup. What do they look like, what can we get out of them, what relevance can we perceive amongst them?

2.1.2 K-regular Graphs

Firstly, let us recall a definition; we assume that the reader is familiar with the basic notions of graph theory, such as bipartite graphs, connected components, reachability, etc..

Definition 2.1.2 (k-regular)**.** *A graph G is called k-regular just when every vertex $u \in G$ has exactly k neighbours.*

There is a connection between the most general solutions to the Pup and k-regular unit graphs. Having k-regular unit graphs in solutions means that we are exploiting the *InterUnitCap* capacity for connections between units to the fullest. In a sense, k-regular unit graphs are thus the most general solutions.

In the case where $k = 2$, there is exactly one k-regular graph; the cycle. In the case where k is odd, k-regular unit graphs only exist if there is an even number of units ("hand-shaking lemma"). Moreover, for $k > 2$ the number of distinct most general unit graphs grows rapidly: for instance, for $k = 4$ there are six distinct graphs on eight vertices, and 8037418 on sixteen vertices; for twenty vertices not all distinct graphs have been constructed [30].

It can be shown, though, that all these solution topologies can be enforced. We include here another observation from [3].

Observation 2.1.3 (PUP Instances and k-regular Graphs [3])**.** *For every k-regular graph G_k there exists a PUP instance $G = (S, Z, E)$ with InterUnitCap $= k$ such that in every solution of G the unit graph is G_k.*

Which also leads to the following corollary.

Corollary 2.1.4. *If there exists a solution to a PUP instance $G = (S, Z, E)$ then there exists a solution to the PUP instance of every subgraph H of G.*

What is also clear from the above Observation 2.1.3 is that as the *InterUnitCap* values increase, if there are no special restrictions on the solution topology of the application domain, then it is practically not feasible to iteratively try all most general solution topologies, so a brute force algorithm would "blow-up" exponentially. Hence, the solution topology will have to be inferred instead.

2.1.3 Forbidden Subgraphs of the PUP

Another known result from [3] regarding the PUP instances is the following.

Lemma 2.1.5 (Forbidden Subgraphs of the PUP [3])**.** *A PUP instance has no solution if it contains $K_{1,n}$ or $K_{n,1}$ as a subgraph, where $n = ((\mathit{InterUnitCap}+1) * \mathit{UnitCap}) + 1$.*

The intuition behind this lemma is the following. Every sensor (or zone) in the input graph has a maximum degree because there are so many zones (or sensors) that can be placed in partner units around the unit that holds the initial sensor, in order to maintain the minimum distance of 1 that they had in the input graph. The threshold of the lemma is explained as follows. Each unit can be connected with, up to, *InterUnitCap* other partner units and each of all these connected units ($IUC + 1$) can hold up to *UnitCap* sensors and zones. So, $(\mathit{InterUnitCap} + 1) * \mathit{UnitCap})$ is the maximum degree of any vertex of the input graph in order for the instance to be solvable.

2.1.4 Bounds on the Number of Units Needed

In previous work [3], it has been shown that the number of units in a solution is bounded from below by $lb = \lceil \frac{\max(|\mathcal{V}_1|,|\mathcal{V}_2|)}{UnitCap} \rceil$, since then at least every unit but one will be have reached their full capacity and also that the the amount of units needed is bounded from above by, the obvious, $ub = |\mathcal{V}_1| + |\mathcal{V}_2|$, otherwise there would be empty units in the solution. Lastly, they noted that if we have more than one connected components each with an upper bound of ub_i, then we can safely state $ub = \sum ub_i$.

Next, we are going to present yet another result from [3]; the stronger upper bound $ub = \max(|\mathcal{V}_1|, |\mathcal{V}_2|)$ holds if $UnitCap > 1$ and $InterUnitCap = 2$. The instance depicted in figure 2.1 shows that this ub does not hold for $UnitCap = 1$ and $InterUnitCap = 2$.

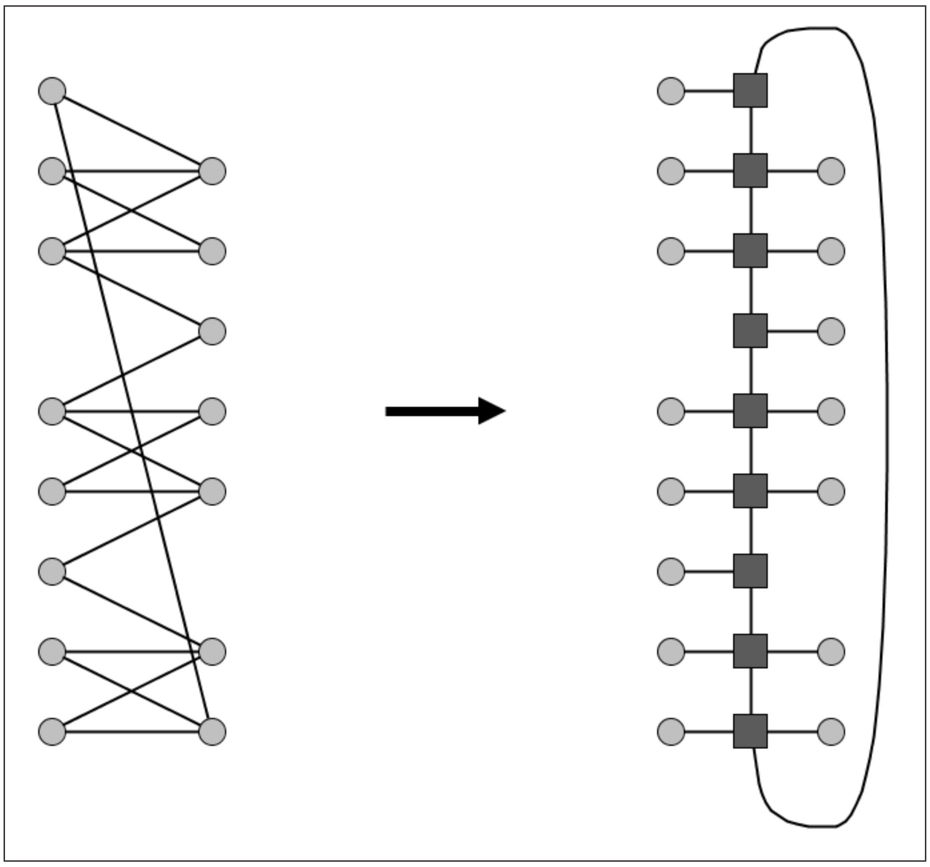

Figure 2.1: An instance requiring more than $\max(|\mathcal{V}_1|, |\mathcal{V}_2|)$ units

It is constructed starting from the solution shown; intuitively, it enforces a "hole" in the units on the left-hand side that can not be closed without violating one of the partner unit constraints.

We now state the above result more formally.

Proposition 2.1.6 (Upper Bound for *UnitCap* > 1 and *InterUnitCap* $= 2$ [3])**.** *For a* PUP*-instance* $G = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$ *with UnitCap* > 1 *and InterUnitCap* $= 2$ *there exists a solution if and only if there exists a solution that uses at most* $ub = \max(|\mathcal{V}_1|, |\mathcal{V}_2|)$ *units.*

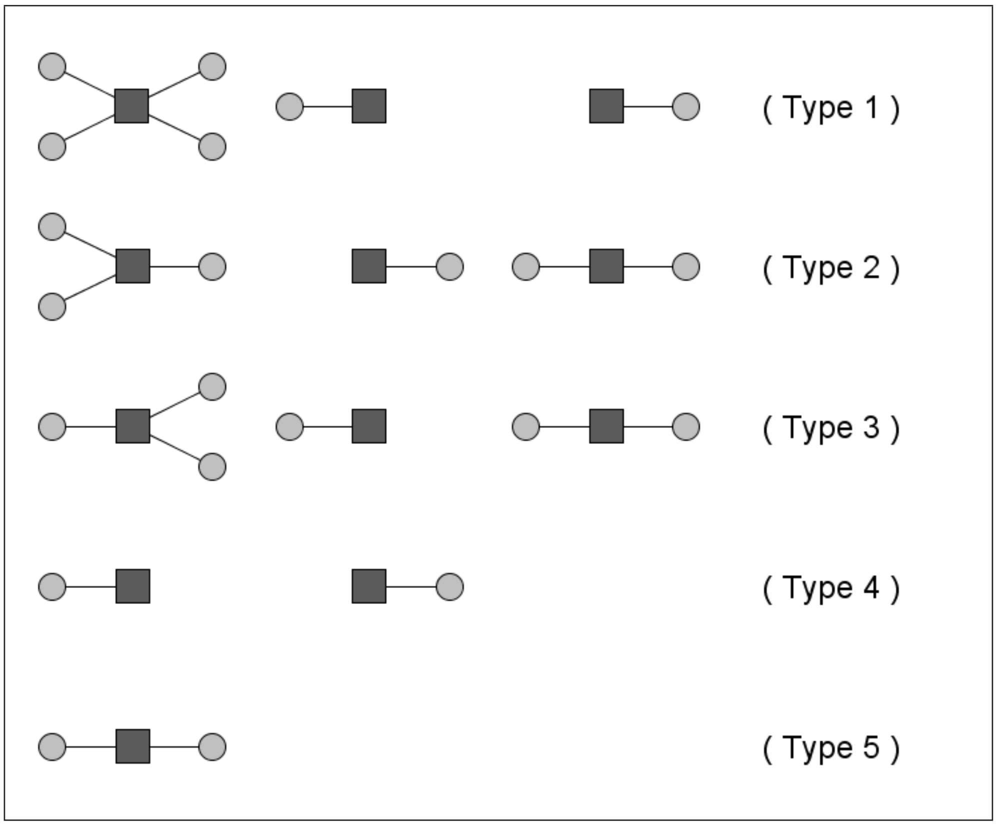

Figure 2.2: The different types of solution building-blocks for *UnitCap* $= 2$

Note that this upper bound is not tight - by doing a proper case analysis it is certainly possible to obtain an upper bound that depends on the value of *UnitCap*, and takes into account the possibility of merging more than two units at a time.

Finally, we would like to notice that for the above mentioned case a tighter bound for the general PUP remains to be computed.

The importance of knowing a better upper bound on the number of units needed for a solution is outlined later in the chapter.

2.2 The Case of *InterUnitCap* $= 2$

We will introduce the Special Partner Units Problem (SPUP); i.e., the specific version of the PUP where *InterUnitCap* is equal to exactly 2. This version of the PUP is of high industrial importance as it has many applications in many of the areas presented in the introduction. We will see that the SPup is tractable, and after going through its properties we will state the NLogSpace search algorithm presented in [3].

For ease of presentation in the sequel we make the same simplifying assumption used in [3]; that the underlying bipartite graph is connected. This does not affect solutions of the decision and the search versions of the Pup, where the connected components can be tackled independently. Though, for optimal solutions the connected components of an underlying graph will have to be considered simultaneously; cf. the discussion in section 2.2.4.

2.2.1 Path Decompositions

The algorithm in [3] decides the SPup by exploiting the notion of path decomposition of a certain width, of a given graph. We next recall the respective definitions.

Definition 2.2.1 (Path Decomposition [3])**.** *A path decomposition of a graph $G = (\mathcal{V}, \mathcal{E})$ is a pair (P, χ) such that P is a simple path; i.e., P does not contain cycles. The function χ associates to every vertex W of the path P a subset $\mathcal{B} \subseteq \mathcal{V}$ such that*

(i). for every vertex V in $\mathcal{V}$ there is a vertex W on P with $V \in \chi(W)$;

(ii). for every edge (V_1, V_2) in $\mathcal{E}$ there is a vertex W with $\{V_1, V_2\} \subseteq \chi(w)$; and

(iii). for every vertex V in $\mathcal{V}$ the set $\{W \mid V \in \chi(W)\}$ induces a subpath of P.

Condition (iii) is called the connectedness condition. The subsets $\mathcal{B}$ associated with the path vertices are called bags*. The* width *of a path decomposition is* $\max_W(|\chi(W) - 1|)$*. The* pathwidth *$pw(G)$ of a graph is the minimum width over all its path decompositions.*

2.2.2 Basic Properties of the SPup

We proceed by identifying the basic properties of the SPup. The key observation here is that the units and their interconnections form a special kind of unit graph in any solution; either a simple path, or a simple cycle. This holds because each unit is connected to at most two partner units.

Moreover, cycles are more general unit graphs than paths; every solution can be extended to a cyclic solution and hence in the sequel we only consider cyclic solutions.

Exploiting this observation, we can transform the SPup into the problem of finding a suitable path decomposition of the input graph G:

Theorem 2.2.2 (SPup is Path-Decomposable [3]). *Assume a* SPup *instance given by a graph $G = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$ is solvable with a solution graph G^* with $|\mathcal{U}| = n$. Let f be the unit function that associates vertices from G to $\mathcal{U}$. Then there is a path decomposition (P, χ) of G of pathwidth $\leq (3 * 2 * UnitCap) - 1$, with the following special properties:*

(i) The length of P is $n-1$; $P = W_1, \ldots W_{n-1}$.

(ii) There are sets $\mathcal{S}_1 \subseteq \mathcal{V}_1$, $\mathcal{S}_2 \subseteq \mathcal{V}_2$ with $|\mathcal{S}_i| \leq UnitCap$ such that $\mathcal{S}_1 \cup \mathcal{S}_2$ are in every bag of the path decomposition.

(iii) Apart from $\mathcal{S}_1 \cup \mathcal{S}_2$ each bag contains at most $2 * UnitCap$ elements from $\mathcal{V}_1$ (or $\mathcal{V}_2$, respectively).

(iv) For any vertex $V \in \mathcal{V}_1 \cup \mathcal{V}_2$ all neighbours of V appear in three consecutive bags of the path decomposition (assuming the first and last bag to be connected).

(v) For each bag $\chi(W_i)$ of it holds $\chi(W_i) = f^{-1}(U_1) \cup f^{-1}(U_i) \cup f^{-1}(U_{i+1})$ for $1 \leq i \leq n-1$.

(vi) $\mathcal{S}_1 = f^{-1}(U_1) \cap \mathcal{V}_1$ and $\mathcal{S}_2 = f^{-1}(U_1) \cap \mathcal{V}_2$.

Intuitively, the vertices in the sets $\mathcal{S}_1$ and $\mathcal{S}_2$ from condition (ii) above are those that close the cycle; i.e., that are connected to unit U_1. These have to be in every bag as some of their neighbours might only appear on the last unit U_n. If all neighbours of $\mathcal{S}_1$ and $\mathcal{S}_2$ already appear in $U_1 \cup U_2$ then we need consider only paths as unit graphs instead of cycles, and the pathwidth is hence decreased by $2 * UnitCap$.

2.2.3 An Algorithm for the SPup

By Theorem 2.2.2 we know that if an SPup instance is solvable then there is a path decomposition, but we still need an algorithm for finding such suitable path decompositions; path decompositions with specific properties:

- The paths should be short (the number of bags reflects the number of units); and hence,
- The bags should be rather full (in "good" solutions the units will be filled up).
- The construction of the bags must be interleaved with checking the additional constraints.

In what follows we give the description of this non-deterministic algorithm as presented by [3] and inspired by [23]. The bags on the path decomposition are guessed. The initial bag partitions the graph into a set of remaining components that are recursively processed simultaneously. A single bag suffices to remember which part of the graph has already been processed; the bag *separates* the processed part of the graph from the remaining components. Consequently, all we have to store is the current bag and the remaining components. It turns out that for this we only need logarithmic space, and thus the algorithm runs in NLogSpace, and hence in polynomial time [13].

In addition to the bags the unit function is guessed, too. According to condition (iv) from above all neighbours of any vertex in G occur in three consecutive bags in the path decomposition. Hence, for checking locally that the unit function is correct it suffices to remember three bags at each step. Finally, we note that the algorithm operates at the level of units rather than bags.

```
DecideSPup($G$) [3]
1  Guess disjoint non-empty $U_1, U_2 \subseteq \mathcal{V}(G)$
   with $|U_i \cap \mathcal{V}_1| \leq \mathit{UnitCap} \geq |U_i \cap \mathcal{V}_2|$
2  $C_R \leftarrow G \setminus (U_1 \cup U_2)$
3  if DecideSPup $(C_R, \langle U_1, U_2\rangle, \langle U_1, U_2\rangle)$
4      then ACCEPT
5      else  REJECT
```

```
DecideSPup($C_R, \langle U_1, U_2\rangle, \langle U_{i-1}, U_i\rangle$) [3]
 1  if $C_R = \emptyset$
 2      then
 3          if $\forall V \in U_1$ nb$(V) \subseteq U_1 \cup U_2 \cup U_i$ and
            $\forall V \in U_i$ nb$(V) \subseteq U_{i-1} \cup U_i \cup U_1$
 4              then ACCEPT
 5              else  REJECT
 6      else
 7          Guess non-empty $U_{i+1} \subseteq \mathcal{V}(\bigcup C_R)$
            with $|U_{i+1} \cap \mathcal{V}_1| \leq \mathit{UnitCap} \geq |U_{i+1} \cap \mathcal{V}_2|$
 8          For $V \in U_i$ check nb$(V) \subseteq (U_{i-1} \cup U_i \cup U_{i+1})$
 9          $C'_R \leftarrow (C_R \setminus U_{i+1})$
10          DecideSPup $(C'_R, \langle U_1, U_2\rangle, \langle U_i, U_{i+1}\rangle)$
```

We next make a number of observations that were exploited to turn DecideSPup into a practically efficient algorithm.

Observation 2.2.3 (Guiding the Guessing [3]). *Not all zones and sensors assigned to units have to be chosen randomly. At most UnitCap neighbours of sensors and zones on the first unit can be assigned to the last unit. Hence the following holds:*[1]

$$|nb_s(U_1) \setminus (U_1 \cup U_2)| \leq \mathit{UnitCap} \geq |nb_z(U_1) \setminus (U_1 \cup U_2)|.$$

Moreover, the neighbours of U_1 not assigned to U_1 or U_2 may only be guessed in the last step, where the number of unprocessed sensors (or zones) is at most UnitCap.

Starting from $i \geq 2$ we have the stronger:

$$(nb_s(U_i) \setminus (U_i \cup U_{i-1})) \subseteq U_{i+1} \supseteq (nb_z(U_i) \setminus (U_i \cup U_{i-1})).$$

Observation 2.2.4 (Finding Optimal Solutions First [3]). *Next recall that "good" solutions correspond to short path decompositions with filled-up bags, and the number of units used in the solution of a* SPUP *instance $G = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$ is bounded by $lb = \lceil \frac{\max(|V_1|, |V_2|)}{\mathit{UnitCap}} \rceil$ from below and by $ub = \max(|V_1|, |V_2|)$ from above. Hence we can apply iterative deepening search: First, try to find a solution with lb units; if that fails increase lb by one; hence the first solution found will be optimal.*

This yields the following corollary.

Corollary 2.2.5 (Tractability of SPUOP [3] [3]). *On connected input graphs the* SPUOP *is solvable in* NLOGSPACE.

Note that branch-and-bound-search (on the number of units used) can not be used: E.g. a $K_{6,6}$ graph does not admit solutions with more than three units.

Observation 2.2.6 (Symmetry Breaking). *We already observed that cycles are more general unit graphs than paths. But with cycles for unit graphs there are two types of rotational symmetry: For a solution with unit graph $V_{U_1}, \ldots, V_{U_n}, V_{U_1}$ there is (1) a solution $V_{U_2}, \ldots, V_{U_n}, V_{U_1}, V_{U_2}$, etc.; in addition there also is (2) the solution $V_{U_1}, V_{U_n}, V_{U_{n-1}}, \ldots, V_{U_2}, V_{U_1}$. We can break these symmetries by stipulating that*

- *the first sensor is assigned to unit U_1; and*
- *the second sensor appears somewhere on the first half of the cycle.*

[1]We denote by $\mathrm{nb}_s(U)$ ($\mathrm{nb}_z(U)$) the set of sensors (zones) that are adjacent in the input graph to zones (sensors) assigned to U.

2.2.4 SPup and Multiple Connected Components

As we have said, the above algorithm works only for input graphs of one connected component. The problem becomes more complex for instances of multiple connected components. Part of the problem is that any two connected components may either have to be assigned to the same, or to two distinct unit graph(s). Accordingly, we are a priori not sure which of the two choices leads to optimal results - for instance, if we assume that $\mathit{UnitCap} = 2$ then two $K_{3,3}$ should be placed on one cyclic unit graph, while a $K_{6,6}$ must stand alone.

2.2.5 Other SPup Approaches

We have shown the PTime algorithm introduced in [3] for solving the SPup for up to logarithmically many connected components. Alternatively, in the same paper by Gottlob et al. [3], quite a few more encodings were introduced for the SPup and the Pup. From these encodings we will focus on the two with the better results[2].

These two are the encodings in the frameworks of propositional satisfiability testing (Sat) and constraint solving (Csp). We will see in the following Chapter how we can try to improve those encodings by stating new results.

Closely related to the performance of the above is the discovery of a tighter upper bound of units needed to compute a solution. As we have seen in Observation 2.2.4, the process used in the DecideSPup algorithm as well as in the encodings just mentioned, is iterative deepening search. That is, to initially check if an optimal solution exists; i.e. a solution using the minimum number of units. And then, if that fails try with the minimum number +1, then +2 etc. up to the upper bound of number of units a solution can have. It is clear that finding a tighter upper bound on this maximum number, will greatly improve the runtime for unsolvable instances — as for them the iteration will have to iterate from the lower bound all the way to the upper bound of the number of units.

Finally, it is our strong belief that the runtime of the above could be greatly decreased by changing this approach to a slightly different one. By analyzing experimental results, we have found out that there is a high probability that the following event occurs. If the instance is solvable, it has an optimal solution or it has a solution using the maximum number of units. The case of an instance not having an optimal or worst solution and yet be solvable, appeared less often. So, in the end, we concluded that an instance that is solvable uses either the minimum or the maximum number

[2]Better in terms of the problem models that were selected by [3]; there is no guarantee that those problem models are the best ones possible though.

of units with higher probability. As a result of that we wish to propose an alternative approach to the current one. This lemma, presents a modified version of the iterative deepening search used in the DecideSPup algorithm and in the encodings in [3].

Lemma 2.2.7 (Modified Iterative Deepening Search)**.** *First, try to find a solution with lower bound units; if that fails try to find a solution with upper bound units; if that fails as well we try to find a solution with lower bound* $+1$ *units; if that fails as well we try to find a solution with lower bound* $+2$ *units, etc..*

Experimental results have not been made to test the efficiency of this; the work on the Pup is ongoing.

Chapter 3

Towards an Efficient Pup Encoding

In this part we present results on both the special and the general version of the Partner Units Problem. Our results on the special version (SPup) aim at improving the effectiveness of the search algorithm and the encodings presented in [3] and also at giving a deeper understanding on the problem itself. We must also note here, as we said before, that the algorithm presented in section 2.2.3 is more a theoretical result aiming at setting the bound of the complexity of the problem, than a result to be used as an actual practical tool. As we discussed, there are other encodings of the SPup and of the Pup that are practically more efficient. In particular, we have said that the problem can be more efficiently solved if encoded in Sat and in Csp all of which can be considered as state-of-the-art for optimization problems [27]. The results presented on the SPup try to improve the runtime of these encodings by imposing new implied constraints or search conditions while others revolve around the theoretical aspect of the problem and the algorithm.

The results that refer to the general case of the Pup are presented as part of the general intuition of the problem as well as for laying the foundations for a possible algorithm to solve this general version. It is intriguing how many different versions the problem has and how complicated its understanding really is. In what follows, we try to offer the reader enough notes on the properties of the problem to help provide a solid understanding and promote future research on it.

3.1 Results on the Pup

We begin by introducing two results on the Pup.

3.1.1 Loose Connectedness Constraint

The first result we are presenting in this section is a constraint that enforces connection relations from the input graph to be maintained in the unit graph as well. In particular, we show that there exists a threshold of common neighbours between any two sensors (zones) that, if surpassed, enforces that the two sensors (units) will have to be "close" in the unit graph.

Proposition 3.1.1 (Loose Connectedness Constraint)**.** *If two sensors (or zones) of the input graph $G = (S, Z, E)$ have a number of common neighbours equal to $(\mathit{InterUnitCap} * \mathit{UnitCap}) + 1$ then in the solution graph they will be in the same unit or in directly connected units.*

Proof. Towards contradiction we have the following. Suppose that the two sensor vertices will not be in the same or directly connected units. That implies that the minimum distance of $s_1 \in U_1$ and $s_2 \in U_2$ in the solution graph will be greater than or qual to 2.

Now, because s_1 and s_2 share $(\mathit{InterUnitCap} * \mathit{UnitCap}) + 1$ common neighbours that means that the $((\mathit{InterUnitCap} * \mathit{UnitCap}) + 1)$ −zones will have to be spread in (at least) two units. Since s_1 and s_2 where directly connected to them in the input graph, they have to be connected (distance 1) in the solution graph. Now, because U_1 and U_2 have distance 2 it means that the only place these zones can go is in the at most $\mathit{InterUnitCap}$−many units between U_1 and U_2. Contradiction! As we said that we have $((\mathit{InterUnitCap} * \mathit{UnitCap}) + 1)$ −zones and all have to go into those units. ■

The importance of this constraint is that it provides a way to structure the unit graph. If this was to be included in an encoding the solver could check these kind of constraints and check for violations of the *UnitCap* and / or the *InterUnitCap*. If that happens, then the instance can be instantly rejected as unsolvable.

3.1.2 Forbidden Fan-Path

The following proposition presents another case of forbidden subgraphs for the Pup. In particular we are interested in the degree sequence of sensors (zones) in acyclic connected subgraphs of the input graph. We show that there is a specific sequence threshold that, if exceeded, guarantees that the instance is unsolvable.

Before going on to the proposition we give the following definition.

Definition 3.1.2 (Fan-path)**.** *Given a* Pup *instance $G = (S, Z, E)$, we call Fan-path an acyclic connected subgraph of G.*

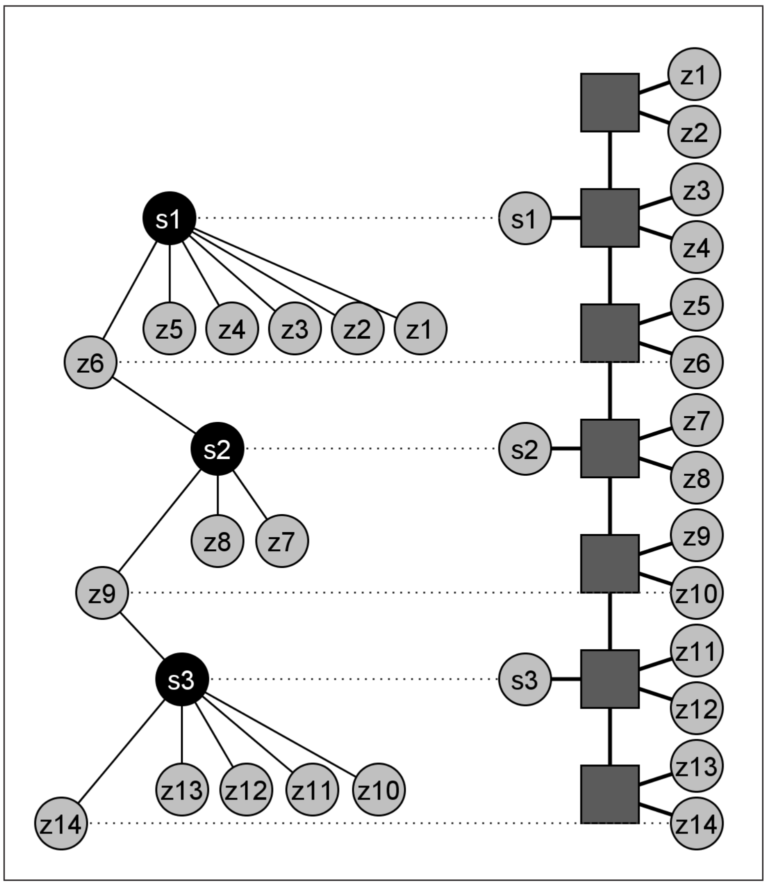

Figure 3.1: An example of a $\mathit{InterUnitCap} = \mathit{UnitCap} = 2 \Rightarrow n = 6$ and $m = 5$ instance of a maximal solvable sequence of $n(m-1)n$.

Proposition 3.1.3 (Forbidden Fan-path). *A* Pup *instance $G = (S, Z, E)$ is unsolvable if there exists a path p of the form:*

$$s_1, z_1, ..., z_k, s_2, z_{k+1}, ..., z_g, s_f \text{ with } s_i \in S \;\forall\; 1 \leq i \leq f \text{ and } z_j \in Z \;\forall\; 1 \leq j \leq g$$

in the Fan-path such that:

- $d(s_1) = n = d(s_f)$*; and*
- $d(s_i) \geq m \;\forall\; 1 < i < n.$

and

- $n = (\mathit{InterUnitCap} + 1) * \mathit{UnitCap}$*; and*
- $m = (\mathit{InterUnitCap} * \mathit{UnitCap}) + 1.$

Proof. Let us assume, without loss of generality, that initially we have a sensor s_1 with degree n. This number, n, is the maximum number of neighbours a sensor (zone) can have. The next sensor s_2 will have exactly one common neighbor with s_1, since we defined them as *neighboring* and also since the subgraph they are in is acyclic, and let z_1 be their connecting zone. The neighbours (degree) of s_2 can be:

- $m <$ neighbours $\leq n$: The instance is unsolvable. It is easy to see that in this case we cannot preserve the distances between the sensors and zones of the input graph to the solution graph.
- $m =$ neighbours: In this case, s_1 and s_2 will have filled up the capacity in all their connected units. Then, if the next sensor s_3 without loss of generality has z_3 in common with s_2, then it cannot have more than m neighbours, nor any other subsequent sensor s_4, s_5, etc., since the connected units will all be full.

■

We now give an example to offer better understanding of this result. Given a Pup instance of $\mathit{InterUnitCap} = \mathit{UnitCap} = 2 \Rightarrow n = 6$ and $m = 5$ where we have $s_1 = 6$ and $s_f = 6$, any other sensor s_i where $1 < i < f$ in the path can have a degree of up to $4 = m - 1$ for the instance to be solvable. In our example, as shown in figure 3.1, the vertices s_1 and $s_{f=3}$ have the maximum number of neighbours and because of that both of them have filled all the $\mathit{InterUnitCap} + 1 = 3$ units that can host their neighbours. This way, the 2 out of 3 units that could host sensor s_2's neighbours, namely the top and the bottom ones, are full — both include one zone from the neighbors of s_1 and s_3 respectively, and each has the zone that they have in common with s_2; in our image z_6 and z_9 — and so s_2 has only one unit to fill with zones that are exclusively his own. So, $\mathit{UnitCap} = 2$ zones of his own and another 2 zones in common with s_1 and s_3 form the maximum degree of s_2, $m - 1 = 4$.

3.2 Results on the SPup

Two more results on the SPup are shown below.

3.2.1 Connectedness Constraint

We will start our approach to the SPup by stating the following proposition - a new constraint on the connectedness of the graph. Much like Proposition 3.1.1, this is a proposition that has to do with maintaining structural properties of the input graph to the final unit graph.

Proposition 3.2.1 (Connectedness Constraint)**.** *If two sensors (or zones) of the input graph $G = (S, Z, E)$ have a number of common neighbours equal to UnitCap$+1$ in a single connected component with more sensors or zones than $4*$UnitCap then in the solution graph they will either be in the same unit or in directly connected units.*

Proof. Towards contradiction we have the following. Suppose that the two sensor vertices will not be in the same or directly connected units. That implies that the minimum distance of $s_1 \in U_1$ and $s_2 \in U_2$ in the solution graph will be greater than or qual to 2.

Now, because s_1 and s_2 share $UC + 1$ common neighbours that means that the $(UC+1)-$zones will have to be spread in (at least) two units. Since s_1 and s_2 where directly connected to them in the input graph, they have to be connected (distance 1) in the solution graph. Now, because U_1 and U_2 have distance 2 it means that the only place these zones can go is in the unit between U_1 and U_2. Contradiction! As we said that we have $(UC + 1)-$zones and all have to go to one unit. ■

This constraint is much similar to Proposition 3.1.1 and the importance of it is mostly explained there. The main difference is that this constraint is believed to have a bigger impact on the runtime of the solvers. The reason is that its condition is not as rare as the one for the equivalent proposition for the general case. We firmly believe that this one can be met, with a high probability, in many instances and hence could greatly improve the efficiency of an SPup encoding.

3.2.2 Cycle Property

The next proposition again imposes a structural property to the unit graph but in a much more abstract way. We analyze input instances that have cycles and understand what that forces onto our unit graph. What we prove is that if the cycle on the input graph is beyond a set threshold then it also enforces a cyclic topology on the unit graph.

Proposition 3.2.2 (Cycle Property)**.** *If the input graph $G = (S, Z, E)$ has a cycle of length $n \geq (4 * UnitCap) + 2$ then the unit graph will have a cycle of length $\lceil \frac{n}{2*UnitCap} \rceil \leq m \leq n$.*

Proof. Let us start this proof by giving some intuition on the number n and the reason it has to be greater than or equal to $(4 * UnitCap) + 2$. Notice that a cycle of length $(4 * UnitCap)$ in the input graph G would, by definition, fill up exactly two units in the unit graph - that happens because any cycle of length k in G contains $\frac{k}{2}$ sensors and $\frac{k}{2}$ zones. So, a cycle of length $(4 * UnitCap) + \varepsilon$ would impose a minimum cycle of three units on the unit graph. The ε is used here after the pure mathematical notation; the next bigger value of $(4 * \mathit{UnitCap})$. Notice that ε cannot be 1 since our input graph is bipartite and so every cycle has to have even length. As such, the smallest value for ε is 2, hence the $(4 * \mathit{UnitCap}) + 2$.

Now let us go through the bounds of m:

- $m \leq n$: This is trivial, each unit, in the minimal case, can have only 1 sensor or zone and it does not make any sense to have any empty units.

- $\lceil \frac{n}{2 * \mathit{UnitCap}} \rceil \leq m$: The lower bound presented is constructed as follows. We start by diving n by two. This holds because, as explained above, n consists of both sensors and zones. We further divide n into distinct units by dividing with $\mathit{UnitCap}$. Finally, we take the smallest integer not less than this $\frac{n}{2 * \mathit{UnitCap}}$.

■

Results 3.2.2, 3.2.1, and 3.1.1 are very interesting because they are some of the few rare constraints that actually impose a specific topology on the unit graphs, an area that we had not managed to cover with other propositions up to now. In general, because most of our constraints are for unsolvability and forbidden instances, it is rather hard to say something about their corresponding unit graphs, due to the nature of the problem. Moreover, it is one more constraint to be taken into consideration when searching if a graph instance is solvable.

Finally, we want to note that we have not tested our results yet and that these are still theoretical speculations. It remains to be seen whether and how much these new constraints can improve the runtimes of the current encodings.

Chapter 4

Pup Versions & Complexity Analysis

4.1 Overview

At this point two things should be clear. The first is that the Pup is in NP. E.g., we have shown that the Pup can be encoded in frameworks of (Sat) and (Csp). In addition, it is rather straightforward that, given a candidate solution for the Pup we can verify it in PTime. The second is that the Pup has many of different instances depending on the values of the *UnitCap* and the *InterUnitCap*. As we have seen throughout this dissertation, each of the different instances that occur for different values on these two parameters produce quite different sub-problems. It is this nature of the Pup that makes it interesting to approach and yet at the same time, complicated to fully understand.

What we can say after working on the problem for this long is that the main decisive factor for the form of the problem is indeed the *InterUnitCap* with the *UnitCap* following up as well. We can observe that the *UnitCap* does change the problem as well, but in general the approach and the rules hold unchanged for its different non-trivial values. For that reason, we have separated the problem into different versions based on the *InterUnitCap* values and not on the values for the *UnitCap*[1].

It is not hard to see that this focus on the *InterUnitCap* produces an intuitive division for the problem; one can easily see that the problem has a totally different structure for

- *InterUnitCap* values of 0 and 1

[1]We will later on in the chapter that there is another distinction that could be made based on the value of the *UnitCap*, but we will not go into detail on it.

- *InterUnitCap* value of 2
- *InterUnitCap* values of 3 or more

Indeed we observe that for the different values of *IUC* we have the following breakdown into distinct cases:

- ***IUC*** ≤ 1: The solution graph will have only isolated units; for the case of 0 it will have single units totally isolated from the rest and for the case of 1 it will only have pairs of units but once again isolated.
- ***IUC*** $= 2$: The solution graph can have isolated units or, in the most dense case, have units connected in a line or a circle.
- ***IUC*** ≥ 3: The solution graph can have any of the previous topologies but can also be interconnected more, forming a complex unit graph. As the value of *IUC* increases the more complex the unit graph gets.

Based on that, the main cases are described on the table 4.1. We further divide the problem, based on whether the *UnitCap* will have a fixed value k or if it will be a free variable given as part of the input.

InterUnitCap	*UnitCap*	Complexity
0	$\star$ [1]	NP-complete
0	k [†]	PTIME [‡]
1	$\star$ [1]	NP-complete [‡]
1	k [†]	PTIME [‡]
2	$\star$ [1]	?
2	k [†]	$O(n^{f(UC)})$
3+	$\star$ [1]	?
3+	k [†]	?

Table 4.1: PUP Instances and their Complexity

This is an analytical case distinction of the PUP with respect to the values of *InterUnitCap*. More versions of the problem could be explored for the different values

[1]Arbitrary *UnitCap* size given as part of the input.

[†]Fixed number for *UnitCap*.

[‡]Results presented in this paper.

of *UnitCap*, for example, $UnitCap = 1$. But we will not be concerned with them in this dissertation. Let us now go through each of these cases separately. Note that unless stated otherwise, the following results hold for input graphs of arbitrary graph families.

4.2 Case-by-case analysis

4.2.1 $InterUnitCap = 0$

The version of the Pup that restricts the value of the *InterUnitCap* to 0 is the simplest version that we can get. The problem then is transformed to a problem quite similar to fixed-parameter BinPacking, the only difference being that in Pup the equivalent of the items of BinPacking are connected components.

The complexity for the two cases of $IUC = 0$ is tackled. The idea behind the solutions here is based on the trivial structure of the unit graph that is produced. As we described earlier, because of the nature of the restriction the *InterUnitCap* enforces on the unit graph, we will get a result of isolated single units.

We start with the result from [3] where by reduction from BinPacking it can be shown that the optimization version of the Pup is intractable when $InterUnitCap = 0$, and *UnitCap* is part of the input.

Theorem 4.2.1 (Intractability of the PuOp IUC=0 [3])**.** *The* PuOp *is* NP*-complete when* $InterUnitCap = 0$*, and UnitCap is part of the input.*

The next result is again on the case of $InterUnitCap = 0$, but this time the *UnitCap* is a fixed constant number k. We show that this problem is tractable and that it belongs in PTime by exhaustive search.

Theorem 4.2.2 (Tractability of the PuOp $IUC = 0$ & $UC = k$)**.** *The* PuOp *is* PTime *when* $InterUnitCap = 0$*, and UnitCap is a fixed constant number* k*.*

Proof. For this proof, we will be calling two solutions equivalent if they are the same up to the ordering of the units, the ordering of the connected components within each bin and the distinguishing of connected components of the same size. We will show that there are polynomially many nonequivalent solutions - we can find the optimal solution by exhaustive search.

The number of configurations for a single unit is at most $UC^{2 \cdot UC}$, even if we distinguished between different orderings of the connected components, therefore a configuration can be specified by the size of each of the at most UC connected components in the unit. The value $UC^{2 \cdot UC}$ might seem rather large, but for our problem it is just a constant.

If we are not attentive enough about only counting nonequivalent solutions, we might come to the conclusion that since a solution uses at most n units, each of which can be in one of at most $UC^{2 \cdot UC}$ configurations, there are at most $UC^{2 \cdot UC^n}$ solutions. This bound is not polynomial and hence not good enough, but we can improve it by recalling that it only matters how many units of each configuration we have, and not in what order they are in.

As such, if x_i are the numbers of units with configuration i, then the nonequivalent solutions correspond to nonnegative integral solutions to the equation

$$x_1 + x_2 + ... + x_{UC^{2 \cdot UC}} \leq n$$

of which there are at most

$$\binom{n + UC^{2 \cdot UC}}{UC^{2 \cdot UC}}$$

as we know from combinatorics. Now, the above bound is at most a polynomial bound of degree $UC^{2 \cdot UC}$; i.e., $\mathcal{O}(n^{UC^{2 \cdot UC}})$

■

4.2.2 *InterUnitCap* = 1

When we restrict the Pup to *InterUnitCap* = 1 then we get a problem quite similar to the one we just examined in section 4.2.1. For *InterUnitCap* = 1 we can have units to be connected to up to one more other partner unit. This can lead to a graph consisting of single and paired units. Because there are not many connections on the units, the problem is still quite similar to the BinPacking problem and is treated similarly.

Again, the complexity for the two cases of *IUC* = 1 is tackled. Once more, the idea behind the solutions is based on the trivial structure of the unit graph that is being produced. This time because of the nature of the restriction that the *InterUnitCap* enforces on the unit graph we will get a result of isolated single units or up to isolated unit pairs.

Our first result here is an extension of Theorem 4.2.1 where once again by reduction from BinPacking it can be shown that the optimization version of the Pup is intractable when *InterUnitCap* = 1, and *UnitCap* is part of the input.

Theorem 4.2.3 (Intractability of the PuOp *IUC*=1)**.** *The* PuOp *is* NP*-complete when InterUnitCap* $= 1$*, and UnitCap is part of the input.*

Proof. We start by noticing that in the best case the connectedness of the units can reach 1 so we will have pairs of units linked rather than individual units that we had up to now. We observe that we can simply consider these pairs to be single units of $(\mathit{UnitCap})\prime = 2 * \mathit{UnitCap}$. Then we can use the exact same idea as above, for these new units.

And so, again, we reduce from BinPacking, given as items $S_1, \ldots, S_n$, binsize b and number of bins k. Notice here that, without loss of generality, we are assuming the binsize b to be an even number. We make a PuOp instance by creating for every i a biclique with S_i zones and S_i sensors, setting $2 * \mathit{UnitCap} = b$ and $\mathit{InterUnitCap} = 0$. A packing with k of fewer bins exists if and only if there exists a solution to the PuOp with $k/2$ or fewer units. Finally note that BinPacking remains NP-complete when all the numbers are expressed in unary [22]. ■

The next result is again on the case of $\mathit{InterUnitCap} = 1$ but this time, as before, the *UnitCap* is a fixed constant number k. We show that this problem is tractable and that it belongs in PTime by reducing to the equivalent version of $\mathit{InterUnitCap} = 0$ that we proved for Theorem 4.2.2 via exhaustive search.

Theorem 4.2.4 (Tractability of the PuOp $IUC = 1$ & $UC = k$)**.** *The* PuOp *is* PTime *when InterUnitCap* $= 1$*, and UnitCap is a fixed constant number* k*.*

Proof. We will show this by reducing to the case of $\mathit{InterUnitCap} = 0$. The reduction comes natural. We can treat the unit pairs that we have in our instance of $IUC = 1$ as units of double the value of *UnitCap* of the case with $IUC = 0$; i.e.,

$$\mathit{UnitCap}_{IUC=1} = 2 * \mathit{UnitCap}_{IUC=0}$$

for the instance of $\mathit{InterUnitCap} = 0$. That way, we just increase the size of the *UnitCap* and other than that treat the problem exactly as we treated the case of $\mathit{InterUnitCap} = 0$. It is clear, that a solution with l or fewer units exists to the case of $IUC = 1$ if and only if there exists a solution to the $IUC = 0$ with $l/2$ or fewer units.

Hence, the Pup with $\mathit{InterUnitCap} = 1$ and *UnitCap* a a fixed constant number k is in PTime. ■

4.2.3 *InterUnitCap* = 2

For values of *InterUnitCap* = 2 the Pup, or SPup as we have introduced it, takes on a whole different structure and becomes less trivial than the previous cases. Now, the units can be connected enough to form paths and if the path is closed, even cycles. This makes it harder as there are more possible combinations for the unit graph and much more ways of including the sensors and zones in them. Accordingly, this version, and the rest of *InterUnitCap* equal or more, are of great industrial interest, as many real-life problems can be modeled to such instances. One should notice here that one of the biggest differences that occur for *InterUnitCap* ≥ 2 is that a single connected component in the input can be stretched out over many units in the solution - unlike the prior cases where it could only stretch over one unit or a pair of them.

For this case, using the algorithm introduced in [3] and presented in section 2.2.3, Gottlob at al. showed the following:

Theorem 4.2.5 (Tractability of SPuDp [3])**.** *The decision problem for the* SPup *is solvable by the algorithm* DecideSPup *in* NLogSpace *for InterUnitCap* = 2 *and any given fixed value of UnitCap.*

Remark 4.2.6. *We should note here that all the previous results refer to arbitrary input graph families. For the case of InterUnitCap* = 2 *the following classification exists:*

- *The* SPuDp *and the* SPuSp *are in* PTime *for arbitrary input graphs.*
- *The* SPuOp *is in* PTime *only for logarithmically many connected components.*

Remark 4.2.7. *Finally, also note that we still do not know the complexity of this case for when the UnitCap is part of the input and is not a fixed value.*

4.2.4 *InterUnitCap* ≥ 3

No results were proven for these two cases, but we would like to refer the reader to our Conjecture 2.1.1 regarding these results.

4.3 Pup Version Discrepancy

Notice that the Pup is a very multi-layered and a deep problem - for small changes in just one of its parameters cause the problem, and its difficulty, to change completely. As we go deeper and deeper into the different versions, conditions and constraints of the problem, we learn more and yet more questions emerge. There are still a lot of open questions on the complexity of the problem for values of *InterUnitCap* of 2 and more and we invite the reader to independently explore those values.

Finally, recall that we have not examined the possible different sub-versions of the Pup for different values of the other parameter, *UnitCap*. We believe that the three main versions of the problem that we have described ($IUC < 2, IUC = 2$ and $IUC > 2$) can be divided once more in terms of *UnitCap*. In particular, it is our belief that there is one major division for $\mathit{UnitCap} = 1$ and $\mathit{UnitCap} > 1$. We will not analyze it in this dissertation though.

Chapter 5

A Logical Approach to the Pup

5.1 Introduction

It is quite common to come across NP-complete problems on graphs that when restricted to specific graph families can be solved in PTime or even linear-time. This research area started in the 1970s when a lot of papers where written describing PTime algorithms for NP-complete graph problems that only worked for graph families of trees or series-parallel graphs. One characterization of problems that were decidable in linear time on series-parallel graphs was presented by Takamizawa, Nishizeki and Saito in [34]. Research then turned to finding the best generalization of the notion of series-parallelness.

As it turned out, there are numerous graph classes whose structure is fundamentally related to that of trees. In fact one can use this observation to solve problems for which a solution is not known or is a lot harder for arbitrary graphs. Robertson and Seymour pioneered this research writing a significant series of papers on graph minors. It was they who in [31] introduced the notion of *treewidth* for a graph. Intuitively, treewidth is a measure of how close a graph is to a tree. The formal definition will be given later in the chapter.

The concept of treewidth and other related ones were widely applied in the design of (theoretically) efficient algorithms (e.g., Johnson in [29]). Several essential graph classes have a universal bound for the treewidth of their graph members. For example, trees and forests have a treewidth ≤ 1, series-parallel graphs and outerplanar graphs ≤ 2, almost trees with k additional edges have a treewidth of $\leq k+1$, Halin graphs are at ≤ 3 and members of k-terminal recursive graph families[1] have a treewidth of

[1] The statement assumes that all the basis graphs in the definition, with edges added between all pairs of terminals, also have treewidth $\leq k$.

$\leq k$. More detailed analysis on these families and classes can be found in [12, 28].

Concurrently, research lead to similar measures of graphs, some of which were suggested as a generalization of the notion of series-parallel. And so, also in this case linear time algorithms were found for these graph classes and for other previously known classes of graphs. A notable suggested measure was the notion of tree-partite graph by Seese in [33], which is closely linked to the treewidth concept.

There are many ways of finding a suitable "template" for the design of linear time algorithms for various problems on these graphs, from giving a list of examples with informal principles [2, 6] to definition of a language in which the property must be expressed [7, 14, 33] as well as some combined approaches [35]. In this work, we will use the second approach - definition of a language in which the property must be expressed. The language we will be using is the *Monadic Second Order Logic* (MSOL).

Monadic second order logic is a very powerful expressive language which contains the propositional logic connectives $\wedge$, $\neg$, $\vee$, $\rightarrow$ and $\leftrightarrow$ (with the usual meanings: and, not, or, implies, if and only if, respectively), individual variables $x, y, z, x_1, x_2, ...,$ existential ($\exists$) and universal ($\forall$) quantifiers and predicates. Moreover, it is distinguished from first-order logic by the following additions: it contains set variables $X, Y, Z, U, X_1, X_2, ...,$ the membership symbol $\in$, and it allows existential and universal quantification over set variables. Mathematical logic has traditionally been concerned with the "intrinsic validity" or "satisfiability" of statements in the form of logical formulae. In order to do so, one must introduced structures as representatives of possible worlds, and the satisfaction relation $\models$ between a structure and a formula. A structure is a set (of objects in the universe) and, for every predicate symbol in the logical language under consideration, a table of tuples of objects satisfying it. In order to show that a formula is "intrinsically valid", a logician must directly or indirectly show that the formula is satisfied by every structure compatible with the logical language, and in order to show that it is satisfiable he must show that it is satisfied by some structure.

Instead, we will use monadic second order logic sentences to define problems on finite graphs. Solving a problem for a given graph will then correspond to deciding if a given finite structure (that represents the graph) satisfies a given sentence (that defines the problem). As an example, the property that a subgraph of G induced by a set Z is connected can be stated as:

$$\begin{gathered}\text{Partition2}(U,V,Z) \equiv (Z = U \cup V) \wedge (U \cap V = \varnothing) \wedge (U \neq Z) \wedge (V \neq Z)\\ \text{Adjacent}(U,V) \equiv \exists u \exists v (v \in V \ \wedge \ u \in U \wedge \ \text{Adj}(u,v))\\ \text{Connected}(Z) \equiv \text{Partition2}(U,V,Z) \rightarrow \text{Adjacent}(U,V)\end{gathered}$$

where lower case variables range over vertices and upper case variables range over vertex sets, and *Adj* is the adjacency relation of the graph and all free variables are considered to have universal quantifiers.

Similarly, in this chapter we will focus on how the PUP can be exploited in this notion - i.e. restricting the input graph families and solving the problem through a logical language: the monadic second order logic. We will start the chapter by giving a formal definition of all the necessary notions that we will be needing and then state the theorems on which we will base our results. In particular we will use the original results from Courcelle [17] and Arnborg, Lagergren and Seese [1].

5.2 Treewidth

We now give a definition of treewidth along with some other definitions that we will be using later on, as presented in [4].

"Tree decompositions [31, 9] and their variants and generalizations [26] constitute a significant success story of Theoretical Computer Science. In fact, tree decompositions and polynomial algorithms for bounded treewidth constitute one of the more effective weapons to attack NP-hard problems, namely, by recognizing and efficiently solving large classes of tractable problem instances. Structural problem decompositions such as treewidth are thus closely related to fixed-parameter tractability [18, 25].

Definition 5.2.1 (Tree Decomposition [4])**.** *A* tree decomposition *of a graph $G = (V, E)$ is a pair $\mathcal{P} = (T, \chi)$ such that $T = (W, F)$ is a tree or forest, and where the function χ associate to every $w \in W$ a subset $B \subseteq V$ such that:*

(i). for every vertex $v \in V$ there is a vertex $w \in W$ with $v \in \chi(w)$;

(ii). for every edge $(v_1, v_2) \in E$ there is a vertex $w \in W$ with $\{v_1, v_2\} \subseteq \chi(w)$; and

(iii). for every vertex $v \in V$ the set $\{w \in W \mid v \in \chi(w)\}$ induces a subtree of T.

Condition (iii) *is called the connectedness condition. The subsets B that are associated with the vertices of W are called bags.*

Definition 5.2.2 (Width, Treewidth [4])**.** *The* width *of a tree decomposition is* $\max_{w \in W}(|\chi(w)| - 1)$*. The* treewidth *$tw(G)$ of G is the minimum width over all of its tree decompositions.*

Definition 5.2.3 (Path Decomposition, Pathwidth [4])**.** *A* path decomposition *of a graph is a tree decomposition where $T = (W, F)$ actually consists of a simple root-leaf path. The* pathwidth *$pw(G)$ of a graph is the minimum width over all its path decompositions.*

The notion of treewidth is easily generalized from graphs to finite structures.

Definition 5.2.4 (Finite Structures [4])**.** *A* finite structure $\mathcal{A}$ *consists of a domain A, and relations $R_1, R_2, ..., R_k$ of arities $a_1, a_2, ..., a_k$, respectively. Each such relation R_i consists of a set of tuples $(r_1, r_2, ..., r_{a_i}) \in R$ where $r_1, r_2, ..., r_{a_i} \in A$.*

A graph $G = (V, E)$ corresponds to a finite structure whose domain is V, with a binary relation E encoding the edges. If G is undirected, then E contain both pairs (v, w) and (w, v) for each edge $\{v, w\}$ of G. Undirected graphs are thus represented as special cases of arbitrary (possibly directed) graphs.

Another related definition is the following.

Definition 5.2.5 (Gaifman Graph [4])**.** *The* Gaifman graph *of a structure $\mathcal{A}$, is the undirected graph $G(\mathcal{A})$ whose set of vertices is the domain A of $\mathcal{A}$ and where there is an edge $\{a, b\}$ if and only if $a, b \in A$ and $a \neq b$, and there exists a tuple in one of the relations of $\mathcal{A}$ in which a and b occur jointly.*

A *tree decomposition of the structure* $\mathcal{A}$ is a tree decomposition for the Gaifman graph, $G(\mathcal{A})$. The *treewidth $tw(\mathcal{A})$ of a structure* $\mathcal{A}$ is defined accordingly, i.e., $\mathrm{tw}(\mathcal{A}) = \mathrm{tw}\,(G(\mathcal{A}))$. Similarly, the pathwidth $\mathrm{pw}(\mathcal{A})$ is defined as $\mathrm{pw}\,(G(\mathcal{A}))$.

The treewidth $\mathrm{tw}(C)$ (pathwidth $\mathrm{pw}(C)$) of a class C of finite structures is the maximum over all $\mathrm{tw}(\mathcal{A})$ ($\mathrm{pw}(\mathcal{A})$) for $\mathcal{A} \in C$. A tree decomposition of a hypergraph H is a tree decomposition of the *primal graph* $G(H)$ of H, which has the same vertices as H and an edge between each pair of vertices that jointly occur in a hyperedge of H.

It is NP-hard to determine the treewidth of a structure $\mathcal{A}$. However, for each fixed k, checking whether $\mathrm{tw}(\mathcal{A}) \leq k$, and if so, computing a tree decomposition for $\mathcal{A}$ of optimal width, is achievable in linear time [8], and was recently shown to be achievable in logarithmic space [19]. Even though the multiplicative constant factor of Bodlanders linear algorithm [8] is exponential in k, there are algorithms that find exact tree decompositions in reasonable time or good upper approximations in many cases of practical relevance; see for example [10, 11] and the references therein.

The treewidth of a graph or relational structure is an invariant that can be used as a parameter to define infinite classes of graphs (or structures) related to problem instances. Many NP-hard problems of practical relevance can be solved in polynomial time on instances of bounded treewidth and some are actually fixed-parameter tractable with respect to treewidth. Given that bounded pathwidth implies bounded treewidth, these results hold a fortiori for bounded pathwidth. The notion of treewidth is at the base of strong meta-theorems such as Courcelles Theorem [17] which we will discuss later in this chapter."

5.3 Monadic Second Order Logic (MSOL)

Monadic second order logic, as we said, contains the propositional logic connectives $\wedge$, $\neg$, $\vee$, $\rightarrow$ and $\leftrightarrow$ (with the usual meanings: and, not, or, implies, if and only if, respectively), individual variables $x, y, z, x_1, x_2, ...$, existential ($\exists$) and universal ($\forall$) quantifiers and predicates. Moreover, it is distinguished from first-order logic by the following additions: it contains set variables $X, Y, Z, U, X_1, X_2, ...$, the membership symbol $\in$, and it allows existential and universal quantification over set variables. Also, it is a restriction of second order logic (SOL) by forcing all relation variables to be unary, i.e., range over sets.

This standard version of MSOL is also referred to as MSO$_1$. Courcelle [16] and other authors also considered an extended version of MSOL called MSO$_2$ which was originally defined for graph input structures only and which extends MSO$_1$ by the possibility of quantifying over subsets of the edges of the input graph. The definition of MSO$_2$ can be easily generalized to arbitrary structures by allowing quantification over subsets of the input relation (notice that in the case of a graph, the input relation is just the edge relation). Thus, for example, if a relational symbol R is part of the signature, the a subformula $(\exists X \subseteq R)\phi(X)$, expressing that there exists a subset X of the relation R such that $\phi(X)$ holds for some formula ϕ, could be part of an MSO$_2$ formula.

5.4 Logic Meta-Theorems

5.4.1 Courcelle's Theorem

Courcelle showed the following fundamental result relating MSOL and problems of a class C of structures of bounded treewidth.

Theorem 5.4.1 ([17]). *For a fixed* MSO$_2$*-sentence ϕ, and a fixed constant k, deciding whether ϕ holds for input structures from a class C is solvable in linear time if the treewidth of C is bounded by k.*

Courcelle‘s celebrated theorem, 5.4.1, essentially states that every problem definable in MSOL can be solved in linear time on graphs of bounded treewidth. However, the multiplicative constants in the running time, which depend on the treewidth and the MSO formula, can be extremely large [21]. Also, since MSO$_1$ is a subset of MSO$_2$, the above theorem obviously also holds for MSO$_1$ formulas.

This result has been generalized by Arnborg, Lagergren, and Seese to *Extended Monadic Second Order Logic* (eMSOL) [1], and by Courcelle and Mosbah to Monadic

Second order evaluations using semiring homomorphisms [15]. In both cases, an MSOL formula with free set variables is used to describe a property, and satisfying assignments to these set variables are evaluated in an appropriate way.

5.4.2 Arnborg, Lagergren, and Seese's Theorem

In the Arnborg et al. version of MSOL, namely eMSOL, one can express optimization and counting problems, a fact that make eMSOL more expressive even from MSO_2. They found that solving problems expressible in this form over input structures is FPT with respect to the treewidth of these input structures.

While, in the original setting in [1], eMSOL properties were defined in a much more general context, for this work it will be enough to state a drastically simplified definition and, accordingly, a simplified master theorem (5.4.3) as per the simplification originally presented in [24].

By optimization we here understand the search for a minimal solution (for our problem) based on some cardinality criteria. The solution itself is an "optimal" assignment of sets to second-order variables that freely occur in some MSOL formula, such that the formula is satisfied over a given input structure. More precisely we state the following definition.

Definition 5.4.2 (Simplified version of a Definition in [1], originally presented by [24])**.** *An optimization problem in an extended MSOL cardinality optimization problem if it can be expressed in the following form. The input of the problem is a structure $\mathcal{A} = (V, E, H)$, where V is a set (the universe of $\mathcal{A}$), and E and H are binary relations over elements of V. Let $\varphi(X)$ be a fixed* MSO_1 *or* MSO_2 *formula over $\mathcal{A}$, where X is either a free set variable ranging over subsets of V or a binary relation variable ranging over subsets of E. The task is to find an assignment among all possible assignments z' to the variable X such that*[2]*:*

$$|z(X)| = \mathrm{opt}\{|z'(X)| : (\mathcal{A}, z') \models \varphi(X)\},$$

where opt *is either min or max. A suitable assignment z is called a solution to the eMSOL cardinality optimization problem.*

Using this definition, Arnborg, Lagergren and Seese found the following principal fixed-parameter tractability result.

[2]An assignment z to the variable X is an interpretation of X that maps X to a subset $z(X)$ of V if X is unary and a subset of E if X is binary.

Theorem 5.4.3 (see [1]). *Solving extended MSOL cardinality problems is fixed-parameter tractable with respect to the treewidth if the input structure. In particular, for a fixed extended MSOL cardinality optimization problem P, and a class C of input structures whose treewidth is bounded by some constant, the following can be done in linear time:*

- *Determine whether P has a solution for an input from C.*
- *Compute a solution for an input from C, if one exists.*

5.5 Pup and MSOL

5.5.1 Approach

After this extensive introduction and background presentation on the field of logical approaches into algorithmic graph problems, one can only expect to see how this can be applied onto the problem at hand - the Pup. The aim of the author is to relate these results to the Pup and come up with new complexity results.

Our approach aims to implement the meta-theorems on some variation of the initial problem. We now present the altered problem that was constructed for this purpose. We will give intuition on why it still remains an interesting problem both in a theoretical and in an applied scope directly after it.

Definition 5.5.1 (Partner Units Embedding Decision Problem - PuEDp).

INPUT: A bipartite graph $G = (S, Z, E_G)$, a topological graph $T = (S, U, Z, E_T)$ and two numbers $n, m \in \mathbb{N}$ where $n \geq 1$ and $m \geq 0$

TASK: Does there exist a subgraph $H \subseteq T$ such that H is a solution to the Pup instance of input graph $G = (S, Z, E_G)$, $\mathit{UnitCap} = n$ and $\mathit{InterUnitCap} = m$?

This new problem introduced here presents a different situation than the Pup. The question at hand is the following: can we find a solution to the Pup instance presented that "fits" in the topological graph given? In other words, assuming we already have a structure on which we are to create our network, can we find a proper subgraph of that structure that serves as a solution to the Pup?

The PuEDp is a problem that remains a theoretical challenge and a practical tool. We will later show formally that the PuEDp is MSOL expressible. For now we will focus on the importance and practicality of the problem presented.

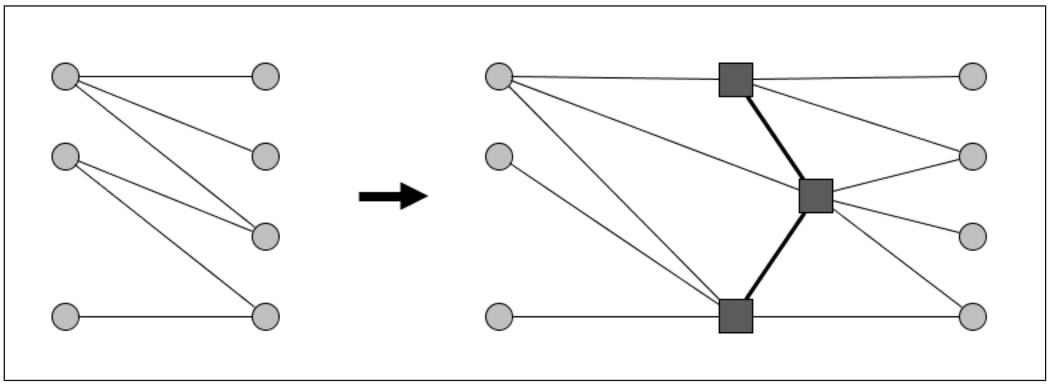

Figure 5.1: An example of an input bipartite graph G and a topological graph T.

On a theoretical scope, this problem is a restriction (or generalization as we will show later) of the Pup and so solving it and working on it provides a better understanding for the actual Pup. Working on the PuEDp gives directions for future work on the area.

It is not rare to have a pre-designed template, network or configuration and want to reuse it. That is where the PuEDp comes in. It is more often than not the case that we have specific configurations and limitations for a design, which in the PuEDp are expressed naturally through the topological graph T. T represents that pre-existing structure that we have to use, the limited resources that we will have to reuse, recycle, etc.. On that notion the PuEDp is a natural extension of the Pup. Initially, one might perceive this as a restriction, but in fact we can re-define the Pup in such a way to make the notion of extension come natural.

Definition 5.5.2 (Partner Units Problem - Pup).

> **INPUT:** A bipartite graph $G = (S, Z, E_G)$ and a topological graph $T_{\max} = (S, U_{\max}, Z, E_{\max})$, where T includes the vertices of S and Z, $U_{\max} = 2{*}\max\{|S|, |Z|\}$ and the edges $E_{\max}$ that form a complete graph for the vertices of $U_{\max}$ and each vertex in S and Z is connected to each vertex in $U_{\max}$ and two numbers $n, m \in \mathbb{N}$ where $n \geq 1$ and $m \geq 0$
>
> **TASK:** Does there exist a subgraph $H \subseteq T$ such that H is a solution to the Pup instance of input graph $G = (S, Z, E_G)$, $\mathit{UnitCap} = n$ and $\mathit{InterUnitCap} = m$?

We can see now that under this new definition, the Pup is only a special case of the PuEDp. It is the case where the PuEDp is equipped with a topological graph that is *maximal*. That means that the set U is maximum and it is fully connected; hence it is a

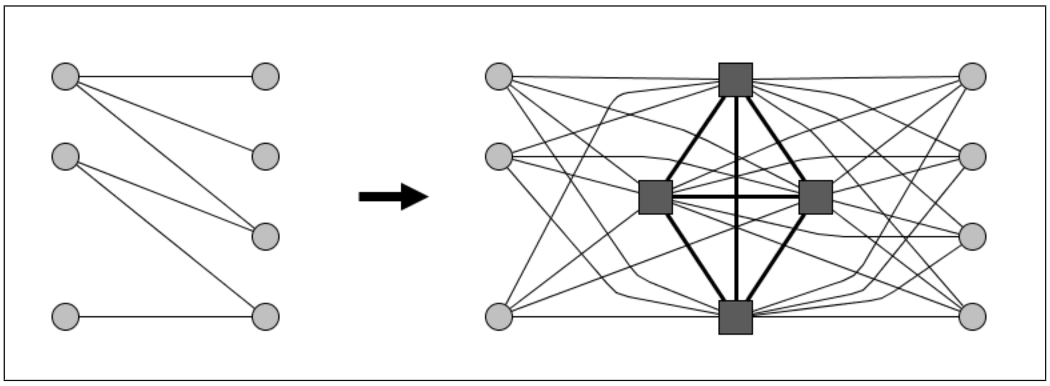

Figure 5.2: An example of an input bipartite graph G and the maximal topological graph T as described in Definition 5.5.2.

complete graph, and for each vertex in S and Z, the edges between that vertex and all the vertices of U exist. It should be intuitive now, that the PuEDp is a generalization of the Pup for *any* topological graph (rather than just the maximal explained above). On a less technical scope, the Pup under this definition is considered a special case of the PuEDp because we are not given a random structure as our base network to work on, but we are actually given the richest structure possible.

5.5.1.1 Decision Problem

Now, we will show that the PuEDp when restricted to *bounded treewidth* input structures G and T, is tractable. We begin by giving the definition of this subversion of the problem.

Definition 5.5.3 (Bounded Partner Units Embedding Decision Problem - BPuEDp).

INPUT: A bipartite graph $G = (S, Z, E_G) \in \mathcal{B}$ and a topological graph $T = (S, U, Z, E_T) \in \mathcal{B}$ where $\mathcal{B} = \{\text{graphs } G \mid \text{tw}(G) \leq k\}$ and two numbers $n, m \in \mathbb{N}$ where $n \geq 1$ and $m \geq 0$

TASK: Does there exist a subgraph $H \subseteq T$ such that H is a solution to the Pup instance of input graph $G = (S, Z, E_G)$, $\mathit{UnitCap} = n$ and $\mathit{InterUnitCap} = m$?

We now state and prove our tractability results.

Theorem 5.5.4 (Tractability of the BPuEDp). *The* BPuEDp *is linear with respect to the treewidth of the input graphs G and T.*

Proof. First, we introduce a finite structure to model our input, that is: $\mathcal{A} = \{S, Z, U, E_G \cup E_T\}$. Note here, that this finite structure has a multi-partite universe, alternatively, this could be presented as $\mathcal{A} = \{A, E\}$ where $A = S \cup Z \cup U$ and $E = E_G \cup E_T$ with added caution in the formulas.

We will say that there exists at least one assignment X that satisfies φ over the structure $\mathcal{A}$ if and only if the BPuEDp is solvable. The BPuEDp is expressed in MSOL, by $\varphi(X)$, as follows:

$$\varphi(X) = \exists su \subseteq E_T\, \exists uz \subseteq E_T\, \exists uu \subseteq E_T\ ((i) \wedge (ii) \wedge \cdots \wedge (vi))$$

Where (i), (ii), ... ,(vi) are defined as follows:

(i). $\forall s \in S\, \exists u \in U\ su(s,u) \wedge \forall u' \in U\ su(s,u') \rightarrow u = u'$;

(ii). $\forall z \in Z\, \exists u \in U\ uz(u,z) \wedge \forall u' \in U\ uz(s,u') \rightarrow u = u'$;

(iii). $\forall s \in S\, \forall z \in Z\ E_G(s,z) \rightarrow \exists u_1, u_2 \in U\ su(s,u_1) \wedge uz(u_2,z) \wedge [(u_1 = u_2) \vee (uu(u_1,u_2))]$;

(iv). $\forall u \in U\, \forall s_1, ..., s_{n+1} \bigwedge_{1 \le i \le n+1} su(s_i,u) \rightarrow \bigvee_{\substack{1 \le i,j \le n+1 \\ i \neq j}} s_i = s_j$;

(v). $\forall u \in U\, \forall z_1, ..., z_{n+1} \bigwedge_{1 \le i \le n+1} uz(u,z_i) \rightarrow \bigvee_{\substack{1 \le i,j \le n+1 \\ i \neq j}} z_i = z_j$; and

(vi). $\forall u \in U\, \forall u_1, ..., u_{m+1} \in U \bigwedge_{1 \le i \le m+1} uu(u,u_i) \rightarrow \bigvee_{\substack{1 \le i,j \le m+1 \\ i \neq j}} u_i = u_j$.

In words, the above predicates are defined as follows:

- $su(x,y)$: sensor x belongs in unit y.
- $uu(x,y)$: unit x is connected to unit y.
- $uz(x,y)$: zone y belongs in unit x.

We show that the $\varphi(X)$ is satisfiable if and only if there exists a subset $H \subseteq T$ such that H is a solution to the Pup instance of input graph $G = (S, Z, E_G)$, $\mathit{UnitCap} = n$ and $\mathit{InterUnitCap} = m$ with respect to the topology of T.

($\Rightarrow$) If $\varphi(X)$ is satisfied then that means that all sensors in S and all zones in Z belong in exactly one unit $u_i \in U$ each. Also, each edge $\{s,z\} \in E_G$ of the input graph has been covered either by including both s and z into a unit u or by including s in u_1, z in u_2, and having u_1 connected to u_2. Finally, we make sure that each unit has up to n zones and up to n sensors (*UnitCap* constraint) by expressions (iv) and (v), and that each unit is connected with up to m partner units (*InterUnitCap* constraint) by expression (vi). Hence, the PUP instance of input graph $G = (S, Z, E_G)$, $UnitCap = n$ and $InterUnitCap = m$ is solvable.

($\Leftarrow$) If the PUP instance of input graph $G = (S, Z, E_G)$, $UnitCap = n$ and $InterUnitCap = m$ is solvable, again with respect to the topology inferred by T, that means the following. There exists a set of units $U = \{u_1, u_2, ..., u_n\}$ such that each unit $u_i \in U$ includes $s_1, ..., s_k$ sensors and/or $z_1, ..., z_l$ zones with $k, l \leq UnitCap$ and each unit $u_i \in U$ of the unit graph has a degree $d(u_i) \leq InterUnitCap$. Therefore, all of the above expressions (i), (ii), ... ,(vi) are satisfied and so there is some assignment X such that $\varphi(X)$ evaluates to true over $\mathcal{A}$.

We have shown that the BPUEDP can be encoded in MSOL, and also, by definition, the input structure has bounded treewidth, therefore by Courcelle's Meta-Theorem, Thm 5.4.1, the BPUEDP is fixed-parameter linear with respect to the treewidth of the input structure. ■

An immediate corollary of Theorem 5.5.4 is the following:

Corollary 5.5.5 (PUEDP is fp-linear). *The* PUEDP *is fixed-parameter linear.*

5.5.1.2 Optimization Problem

We will now extend our previous result for the optimization version of the PUEP. We will show that when the input is restricted to *bounded treewidth* input structures G and T, the PUEOP is tractable as well. We begin by giving the definition of this subversion of the problem.

Definition 5.5.6 (BOUNDED PARTNER UNITS EMBEDDING OPTIMIZATION PROBLEM - BPUEOP).

> **INPUT:** A bipartite graph $G = (S, Z, E_G) \in \mathcal{B}$ and a topological graph $T = (S, U, Z, E_T) \in \mathcal{B}$ where $\mathcal{B} = \{\text{graphs } G \mid \text{tw}(G) \leq k\}$ and two numbers $n, m \in \mathbb{N}$ where $n \geq 1$ and $m \geq 0$
>
> **TASK:** Does there exist a subgraph $H \subseteq T$ such that H is an optimal solution, with respect to the solution space defined by the topological graph T, to the PUP instance of input graph $G = (S, Z, E_G)$, $UnitCap = n$ and $InterUnitCap = m$?

We show the following:

Theorem 5.5.7 (Tractability of the BPUEOP). *The* BPUEOP *is linear with respect to the treewidth of the input graphs G and T.*

Proof. First, we introduce a finite structure to model our input, that is: $\mathcal{A} = \{S, Z, U, E_G \cup E_T\}$ (the same note as in proof of Thm 5.5.4 applies).

The BPuEOp can be expressed as follows as an eMSOL cardinality optimization problem. Find an assignment $z \subseteq U$ to set variable $X = U_{Opt}$ such that:

$$|z(U_{Opt})| = \min\{|z'(U_{Opt})| \, : \langle \mathcal{A}, z' \rangle \models \text{BPuEOp}(U_{Opt})\}$$

Where BPuEOp(U_{Opt}) is an MSO$_2$ expression defined as:

$$\text{BPuEOp}(U_{Opt}) = \exists su \subseteq E_T \, \exists uz \subseteq E_T \, \exists uu \subseteq E_T \, \left((i) \wedge (ii) \wedge \cdots \wedge (vii)\right)$$

Where (i), (ii), ... ,(vii) are defined as follows:

(i). $U_{Opt} \subseteq U$;

(ii). $\forall s \in S \, \exists u \in U_{Opt} \, su(s,u) \wedge \forall u' \in U_{Opt} \, su(s,u') \rightarrow u = u'$;

(iii). $\forall z \in Z \, \exists u \in U_{Opt} \, uz(u,z) \wedge \forall u' \in U_{Opt} \, uz(s,u') \rightarrow u = u'$;

(iv). $\forall s \in S \, \forall z \in Z \, E_G(s,z) \rightarrow \exists u_1, u_2 \in U_{Opt} \, su(s,u_1) \wedge uz(u_2,z) \wedge [(u_1 = u_2) \vee (uu(u_1,u_2))]$;

(v). $\forall u \in U_{Opt} \, \forall s_1, ..., s_{n+1} \bigwedge_{1 \le i \le n+1} su(s_i,u) \rightarrow \bigvee_{\substack{1 \le i,j \le n+1 \\ i \ne j}} s_i = s_j$;

(vi). $\forall u \in U_{Opt} \, \forall z_1, ..., z_{n+1} \bigwedge_{1 \le i \le n+1} uz(u,z_i) \rightarrow \bigvee_{\substack{1 \le i,j \le n+1 \\ i \ne j}} z_i = z_j$; and

(vii). $\forall u \in U_{Opt} \, \forall u_1, ..., u_{m+1} \in U \bigwedge_{1 \le i \le m+1} uu(u,u_i) \rightarrow \bigvee_{\substack{1 \le i,j \le m+1 \\ i \ne j}} u_i = u_j$.

In words, exactly as before, the above predicates are defined as follows:

- $su(x,y)$: sensor x belongs in unit y.
- $uu(x,y)$: unit x is connected to unit y.
- $uz(x,y)$: zone y belongs in unit x.

We show that the BPuEOp(U_{Opt}) is satisfiable if and only if there exists a subset $H \subseteq T$ such that H is the optimal solution, with respect to the solution space defined by the topological graph T, to the Pup instance of input graph $G = (S, Z, E_G)$, *UnitCap* $= n$ and *InterUnitCap* $= m$.

Both directions of this proof are the same as in the proof of Theorem 5.5.4 with the following additions.

($\Rightarrow$) If we have found the minimum number of units that satisfy BPuEOp(U_{Opt}) and that is a solution to the Pup instance of input graph $G = (S, Z, E_G)$, $\mathit{UnitCap} = n$ and $\mathit{InterUnitCap} = m$ then it automatically is the optimal solution with respect to the solution space defined by the topological graph T.

($\Leftarrow$) If the Pup instance of input graph $G = (S, Z, E_G)$, $\mathit{UnitCap} = n$ and $\mathit{InterUnitCap} = m$ is solved optimally, always with respect to the solution space defined by the topological graph T, then that means that the number of units used in BPuEOp(U_{Opt}) is the minimum.

We have shown that the BPuEOp can be encoded as an extended MSOL cardinality problem, and also, by definition, the input structure has bounded treewidth, therefore by Arnborg et al.'s Meta-Theorem, Thm 5.4.3, the BPuEOp is fixed-parameter linear with respect to the treewidth of the input structure.

■

Once more, an immediate corollary of Theorem 5.5.7 is the following:

Corollary 5.5.8 (PuEOp is fp-linear)**.** *The* PuEOp *is fixed-parameter linear.*

5.5.2 An unsolved problem

The original approach we aimed at was to restrict the input graphs of the Pup to the class of bounded treewidth graphs and then express the problem in monadic second order logic or even extended monadic second order logic and apply one of the two meta-theorems presented. This did not have fruitful results. Though the idea of restricting the input graphs to having a bounded treewidth came almost naturally and though these instances were close to the practical input that we get for applications of the problem, MSOL was not expressive enough to encode it. In particular, we had trouble because of the "monadic" nature of MSOL. We found that the problem needs a dyadic second order logic to be encoded (binary relations).

It remains open whether there exists some other extension of MSOL such that the Pup can be encoded in and remain under the range of effect of one of the two meta-theorems proposed.

Chapter 6

Conclusion

The PARTNER UNITS PROBLEM is a complex and multilayered configuration problem. For small value changes in its parameters the problem changes completely. As a result of that,t has numerous versions and they have significant discrepancy. Moreover, we conjecture that almost all of those versions are NP-Complete. A fact that we have proved for two of them.

We started this dissertation with the scientific framework that it belongs to, gave the informal definition and then continued to present the formal definition of the problem. Next, we showed the basic facts and remarks around the PUP that had been noted by previous works and highlighted the research frontier on the PUP by underlining all the crucial previous results done in [3]. We stated the DECIDESPUP algorithm for the special case of the PUP and briefly noted the other encodings that were used in [3].

We continued by presenting four new constraints for the general and for the special case of the problem. Constrains that we believe if included in the two encodings of the problem in SAT and CSP will be able to improve the runtime of these implementations. These results were a product of graph-theoretic analysis on the structural properties of the input graphs and the properties that they enforce to the final solution, a process that not only lead to the results mentioned but also provided crucial intuition for the complexity analysis that continued.

In the next part, we focused on the subdivisions of the problem and their complexity analysis. We started by offering an intuitive division into 3 main categories and then into some subcategories and explained the unique features that shape them. As we described above, the different versions of the problem have many differences between them. We began our complexity analysis with the easiest versions of $\mathit{InterUnitCap} = 0, 1$ and discovered that only they where tractable — i.e., in PTIME. All the rest, that we managed to prove, were in the NP-Complete class.

We used two different approaches to tackle the complexity of these versions. We started with an algorithmic approach and tried to find reductions for the problems, then tried to exploit structural traits with a direct approach and finally we turned to logic.

The final part of our dissertation presents exactly this logical approach. In particular, this direction began with the intent to use the algorithmic meta-theorems of Courcelle [17] and Arnborg at al. [1] to prove the complexity of the remaining cases. Towards this goal, we presented the theory behind this approach and gave a short introduction on the language that we would use; the Monadic Second Order Logic and its extensions. Using this language we found out that it was impossible to express the Pup, but that lead us to the discovery of a new problem; the introduced Partner Units Embedding Decision Problem. This new problem could be considered a generalization of the Pup. Finally, we proved that the PuEDp and the PuEOp are fixed-parameter tractable with respect to the treewidth of their input structures.

6.1 Impact of Work

We have given a thorough introduction to the up-to-date research on the Pup. We introduced four new constraints that give a different perspective on the problem and expand the intuition that we had on the problem. By this addition, we have helped pave the way for more efficient encodings of the Pup and better runtimes. We classified the different versions of the Pup and proved the complexity of more than half of them. We described the difficulties and outlined the reasons for which we could not prove the rest, thus setting the foundations for future research on the complexity analysis. Finally, we introduced a variation of the problem, the PuEp. We defined both decision (PuEDp) and optimization (PuEOp) versions, showed their importance and proved their complexity.

6.2 Open Problems

We believe that our understanding on the Pup is still deficient in many areas and hence there are still open problems related to the Partner Units Problem. We present them here and invite further research on them:

- Creating an improved encoding for the SPup using the constraints introduced in Chapter 3;

- Creating an improved encoding for the PUP using the constrains and search method described in Chapter 3;
- Finding a stronger upper bound on the number of units needed for a solution;
- Determine the complexity of the SPUOP for the case of more than logarithmically many connected components;
- Proving the complexity of the three still open cases;
- Checking whether there is some new extended version of the MSOL that can express the PUP; and
- Finding practical application scenarios and discovering new connections for the two new problems introduced; the (PUEDP) and the (PUEOP).

References

[1] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[2] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23:11–24, April 1989.

[3] M. Aschinger, C. Drescher, G. Gottlob, P. Jeavons, and E. Thorstensen. Tackling the Partner Units Problem. Technical Report RR-10-28, Computing Laboratory, University of Oxford, 2010. Available from the authors.

[4] Markus Aschinger, Conrad Drescher, Georg Gottlob, Peter Jeavons, and Evgenij Thorstensen. Structural Decomposition Methods and What They are Good For. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12–28, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[5] Third International Answer Set Programming Competition 2011. `https://www.mat.unical.it/aspcomp2011/`, 2011.

[6] Marshall W. Bern, Eugene L. Lawler, and A. L. Wong. Linear-time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*.

[7] Hans I. Bodlaender. Dynamic programming on graphs with bounded treewidth, 1987.

[8] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 226–234, New York, NY, USA, 1993. ACM.

[9] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[10] Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. On exact algorithms for treewidth. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 672–683, 2006.

[11] Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations i. upper bounds. *Inf. Comput.*, 208(3):259–275, 2010.

[12] H.L. Bodlaender. Classes of graphs with bounded tree-width. Technical Report RUU-CS-86-22, Department of Information and Computing Sciences, Utrecht University, 1986.

[13] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 4:4–18, 1971.

[14] B. Courcelle. The monadic second-order logic of graphs iii: Treewidth, forbidden minors and complexity issues. *Informatique Thorique*.

[15] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109:49–82, March 1993.

[16] Bruno Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. 1990.

[17] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990.

[18] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer, 1999.

[19] Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152, 2010.

[20] A. Falkner, A. Haselböck, and G. Schenner. Modeling Technical Product Configuration Problems. In *Proceedings of the Workshop on Configuration at ECAI 2010*, Lisbon, Portugal, 2010.

[21] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, LICS '02, pages 215–224, Washington, DC, USA, 2002. IEEE Computer Society.

[22] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman, 1979. p. 226.

[23] G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decomposition and Tractable Queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

[24] Georg Gottlob and Stephanie Tien Lee. A logical approach to multicut problems. *Inf. Process. Lett.*, 103(4):136–141, 2007.

[25] Martin Grohe. Descriptive and parameterized complexity. In J. Flum and M. Rodriguez-Artalejo, editors, *Computer Science Logic CSL'99*, volume 1683, pages 14–31. Springer, 1999.

[26] Petr Hliněný, Sang il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008.

[27] J. N. Hooker. *Integrated Methods for Optimization.* Springer, New York, 2006.

[28] Neil Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

[29] David S. Johnson. The np-completeness column: An ongoing guide. *Journal of Algorithms*, 8(2):285 – 303, 1987.

[30] M. Meringer. Regular Graphs Page.
`http://www.mathe2.uni-bayreuth.de/markus/reggraphs.html`.

[31] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms.*

[32] Daniel Sabin and Rainer Weigel. Product configuration frameworks - a survey. *IEEE Intelligent Systems.*

[33] Detlef Seese. Tree-partite graphs and the complexity of algorithms. In *Fundamentals of Computation Theory*, FCT '85, pages 412–421, London, UK, 1985. Springer-Verlag.

[34] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *J. ACM*, 29:623–641, July 1982.

[35] S.T. Hedetniemi T.V. Wimer and R. Laskar. A methodology for constructing linear graph algorithms. *Congr. Numer:*.