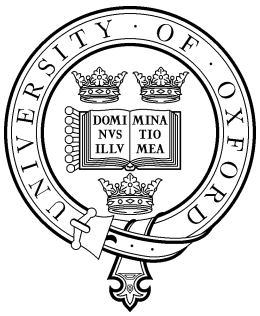
Computing Science

TACKLING THE PARTNER UNITS CONFIGURATION PROBLEM

Markus Aschinger, Conrad Drescher, Gerhard Friedrich, Georg Gottlob,
Peter Jeavons, Anna Ryabokon, Evgenij Thorstensen

CS-RR-10-28



Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Tackling the Partner Units Configuration Problem*

Markus Aschinger†, Conrad Drescher‡, Gerhard Friedrich§, Georg Gottlob¶, Peter Jeavons‖, Anna Ryabokon**, Evgenij Thorstensen††

Abstract

The Partner Units Problem is a type of configuration problem, that frequently occurs in industry. Unfortunately, it can be shown that in the general case (the optimization version of) the problem is intractable. In this technical report we identify a tractable class of problem instances that can be solved by a novel algorithm exploiting the notion of a path decomposition. We also present and evaluate encodings for the general version of the problem in the optimization frameworks of answer set programming, propositional satisfiability testing, constraint solving, and integer programming.

Keywords: Configuration, structural decomposition methods, constraint optimisation, SAT solving, integer programming, answer set programming

*Further copies of this Research Report may be obtained from the Librarian, Oxford University Computing Laboratory, Computing Science, Wolfson Building, Parks Road, Oxford OX1 3QD, England (Telephone: +44-1865-273837, Email: library@comlab.ox.ac.uk).

†Computing Laboratory, University of Oxford

‡Computing Laboratory, University of Oxford

§Institut für Angewandte Informatik, Alpen-Adria-Universität Klagenfurt

¶Computing Laboratory, University of Oxford, Oxford Man Institute of Quantitative Finance

‖Computing Laboratory, University of Oxford

**Institut für Angewandte Informatik, Alpen-Adria-Universität Klagenfurt

††Computing Laboratory, University of Oxford

1 Introduction

The Partner Units Problem (PUP) has recently been proposed as a new benchmark configuration problem [8]. 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 like e.g. intelligent traffic management, or surveillance and security applications. The PUP is used as a novel benchmark instance at the 2011 answer set programming competition [1].

Figure 1 shows a PUP instance and a solution for the case $\mathit{UnitCap} = \mathit{InterUnitCap} = 2$ — 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 technical report 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 e.g. 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 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.

In this work we hence also present encodings in the optimization frameworks of answer set programming, constraint satisfaction, SAT-solving, and integer 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.

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

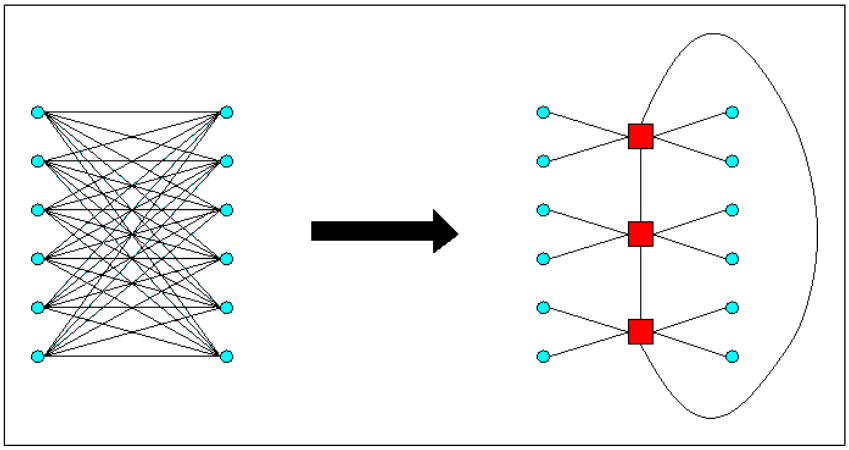


Figure 1: Solving a $K_{6,6}$ Partner Units Instance — Partitioning Sensors and Zones into Units on a Circular Unit Layout

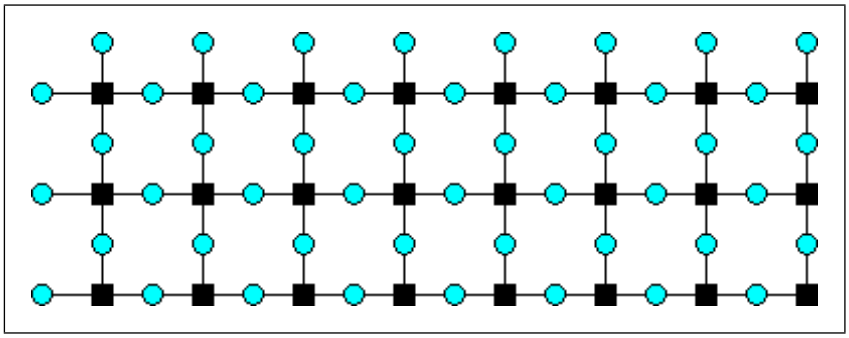


Figure 2: A Grid-like PUP Instance

The remainder of this technical report is organized as follows:

- In section 2 we give the basic formal definitions, and identify general properties of the PUP.
- In section 3 we establish our recent tractability results for the case $\mathit{InterUnitCap} = 2$.
- In section 4 we present problem models in the frameworks of answer set programming, propositional satisfiability testing, integer programming, and constraint solving. These problem models can be used for arbitrary values of *InterUnitCap*, but we also show how they can be adapted to the special case of $\mathit{InterUnitCap} = 2$.
- In section 5 we evaluate the performance.
- In section 6 we conclude.

2 Formal Definition of the Partner Units Problem and Basic Facts

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.

2.1 Formal definition

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.

2.2 The Partner Units Problem is Intractable

By a 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. Observe that clearly the PUP is in NP (cf. also section 4).

Theorem 2.1 (Intractability of the PUOP)**.** *The* PUOP *is* NP*-complete when InterUnitCap* $= 0$*, and UnitCap is part of the input.*

Proof. We reduce from BINPACKING, given as items $S_1, \ldots, S_n$, binsize b and number of bins k. We make a PUOP instance by creating for every i a biclique with S_i zones and S_i sensors, setting *UnitCap* $= b$ and *InterUnitCap* $= 0$. A packing with k of fewer bins exists iff there exists a solution to the PUOP with k or fewer units. Finally note that BINPACKING remains NP-complete when all the numbers are expressed in unary [9]. □

Observe that for practical applications the value of *UnitCap* will not be arbitrary.

2.3 Forbidden Subgraphs of the PUP

It is also not hard to show the following:

Lemma 2.2 (Forbidden Subgraphs of the PUP)**.** *A* PUP *instance has no solution if it contains* $K_{1,n}$ *or* $K_{n,1}$ *as a subgraph, where* $n = ((\textit{InterUnitCap} + 1) * \textit{UnitCap}) + 1$.

Proof. Assume some sensor S has n neighbors, and is assigned to some unit U in a solution. There can be at most *InterUnitCap* neighbors U_{nb} of U, each with at most *UnitCap* zones attached. Hence one of the neighbors of S has to be on a unit different from U and all of the U_{nb}. Contradiction. □

2.4 K-regular Graphs

Next let us discuss the connection between most general solutions to the PUP and k-regular unit graphs: A graph is k-regular if every vertex has exactly k neighbors. 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 thus are most general solutions.

In the case where $k = 2$, there is exactly one k-regular graph, the cycle; we exploit this fact in section 3. 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: E.g. 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 [16]. It can be shown that all these solution topologies can be forced:

Observation 1 (PUP Instances and k-regular Graphs)**.** *For every* k*-regular graph* G_k *there exists a* PUP *instance* G *with* $\textit{InterUnitCap} = k$ *such that in every solution of* G *the unit graph is* G_k.

Proof. Construct the instance G as follows:

1) First connect $2 * \textit{UnitCap}$ vertices (i.e. sensors and zones) to each node in G_k. Let the set of all sensors (zones) be $\mathcal{V}_1$ ($\mathcal{V}_2$).

2) The instance G contains all edges (v_1, v_2), where $v_1 \in \mathcal{V}_1$ and $v_2 \in \mathcal{V}_2$ are connected to either the same or adjacent nodes in G_k.

We show that every solution is isomorphic to G_k. We consider two cases:

- $0 \leq \textit{InterUnitCap} \leq 1$: The result is immediate.
- $k > 1$: Let U_0 be a node in G_k with neighbors, and let $U_j : 1 \leq j \leq k$ be the neighbors. Denote by $\mathcal{V}_1^{U_i}$ and $\mathcal{V}_2^{U_i}$ the sensors and zones created in G for $U_i : 0 \leq i \leq k$. Let G'_k be an optimal solution for G. We need to show two things: (1) For each $0 \leq i \leq k$ both $\mathcal{V}_1^{U_i}$ and $\mathcal{V}_2^{U_i}$ are on the same unit in G'_k. (2) For $0 \leq i < j \leq k$ if $\mathcal{V}_1^{U_i} \cup \mathcal{V}_2^{U_i}$ and $\mathcal{V}_1^{U_j} \cup \mathcal{V}_2^{U_j}$ are connected in G then their units are connected in G'_k. Both (1) and (2) follow from the following: Consider a path of three units U_1, U_2, U_3. Consider the instance constructed by applying steps 1) and 2) to this path. Any solution for this instance has three units. Here

we consider only solutions where the unit graphs are paths, say U_4, U_5, U_6. The sensors and zones originally attached to the middle unit U_2 have to be on the middle unit U_5 — otherwise their $2 * \mathit{UnitCap}$ neighbors won't fit. Moreover, the sensors and zones from U_1 have to be on the same unit, either U_4 or U_6; the sensors and zones from U_3 are on the unit at the other end of the path. With the exception of the cycle of length two, all k-regular unit graphs are composed from such path segments of length three. As every "end" of a path segment is the "middle" of another path segment we did not consider cyclic solutions for the path segments. Conditions (1) and (2) are now easily seen to hold.

□

Hence, if $\mathit{InterUnitCap} > 2$, and there are no restrictions on the solution topology in the application domain, then it is practically not feasible to iteratively try all most general solution topologies. The solution topology will have to be inferred instead.

2.5 An Upper Bound on the Number of Units

In this section we address the question of how many units are necessary (or sufficient, respectively) for finding a solution for a PUP instance $G = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$. These bounds are useful in that they help to limit the depth of iterative deepening search: For finding optimal solutions in most problem models introduced in the sequel we first try to find a solution with lb units; if that fails increase lb by one. Clearly this has the effect that the first solution found will be optimal.

Clearly the number of units in a solution is bounded from below by $lb = \lceil \frac{\max(|\mathcal{V}_1|,|\mathcal{V}_2|)}{\mathit{UnitCap}} \rceil$ — then at least every unit but one is filled to the maximum. For the upper bound we start with the simple observation that $ub = |\mathcal{V}_1| + |\mathcal{V}_2|$ certainly is always enough — otherwise there would be empty units in the solution. Next we observe that if there are multiple connected components in the instance, each with upper bound ub_i, then we can safely set $ub = \sum ub_i$.

Next we are going to show that the stronger upper bound $ub = \max(|\mathcal{V}_1|, |\mathcal{V}_2|)$ holds if $\mathit{UnitCap} > 1$ and $\mathit{InterUnitCap} = 2$. The instance depicted in figure 3 shows that this ub does not hold for $\mathit{UnitCap} = 1$ and $\mathit{InterUnitCap} = 2$. It is constructed starting from the solution shown, using the same construction as in the proof of observation 1. 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.

Proposition 2.3 (Upper Bound for $\mathit{UnitCap} > 1$ and $\mathit{InterUnitCap} = 2$)**.** *For a* PUP*-instance* $G = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E})$ *with* $\mathit{UnitCap} > 1$ *and* $\mathit{InterUnitCap} = 2$ *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.*

Proof. Let $n = \max(|\mathcal{V}_1|, |\mathcal{V}_2|)$, i.e. the upper bound to be proved. Assume there is a solution with $m = n + o$ units. We need to show that then there are at least o pairs of connected units that can be merged. Without loss of generality we may assume that $|\mathcal{V}_1| = |\mathcal{V}_2|$ and $\mathit{UnitCap} = 2$ — if this does not hold the number of mergeable pairs of units can only be greater. We may also assume that the solution is cyclic. A pair of adjacent units can be merged if the sum of the respective sensors and zones respects $\mathit{UnitCap}$.

We distinguish the five types of solution building blocks depicted in figure 4. Let x_i denote the number of units of type i in the solution. In general we have that $m = 3x_1 + 3x_2 + 3x_3 + 2x_4 + x_5$.

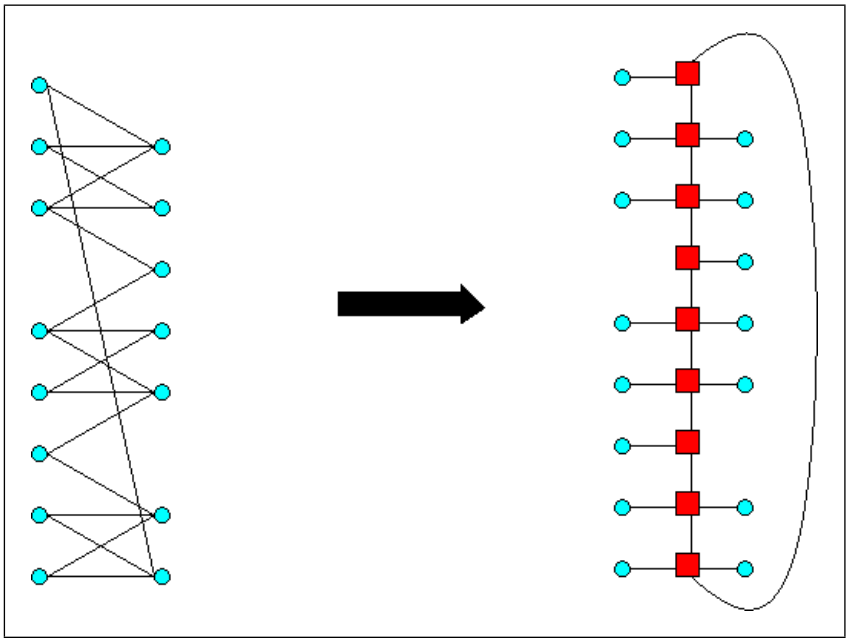


Figure 3: An instance requiring more than $\max(|\mathcal{V}_1|, |\mathcal{V}_2|)$ units

In the simplest case the solution contains only Type 4 and Type 5 units, i.e. $x_4 = 2o$ and $x_5 = n - o$. Here alternatingly picking edges on the cycle yields $\lfloor \frac{m}{2} \rfloor$ pairs of units that can be merged. The maximum number of non-mergeable units is reached for maximum values of x_1. W.l.o.g. we assume that $x_2 = x_3 = x_5 = 0$, i.e. $n - o \mod 3 = 0$, and consequently have that $x_1 = n - o$ and $x_4 = 2o$. The key observation is that for every "full" unit we introduce two mergeable units. The number of "full" units is given by $\frac{n-o}{3}$, and the total number of units that can be blocked from being mergeable is then given by $\frac{n-o}{3} - 1$, i.e. less than the number of units introduced due to full units. □

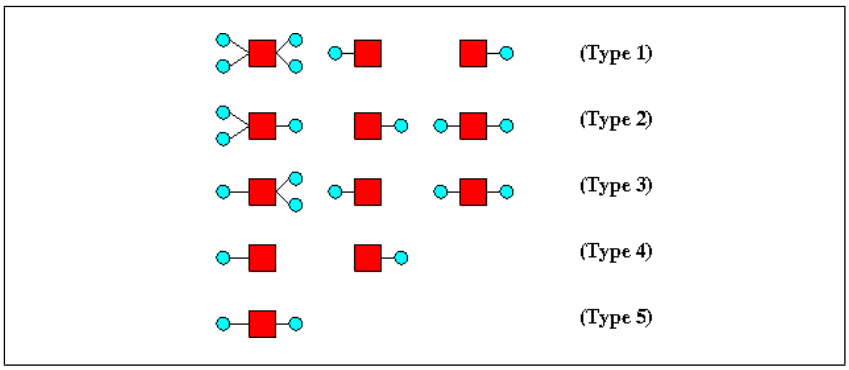


Figure 4: The different types of solution building-blocks for *UnitCap* $= 2$

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.

2.6 Symmetry breaking

If we don't use iterative deepening search then in some problem models there might be empty units. Here we can do symmetry breaking by demanding that whenever unit j has a sensor or zone assigned to it, then every unit $j' < j$ also has some sensor or zone assigned to it.

We can also rule out a quarter of the connections between sensors (or alternatively, zones) and units immediately. Consider sensors and units: Sensor 1 must be somewhere, so it might as well be on unit 1. Sensor 2 can either be on unit 1 or on a new unit, let's say 2. Unfortunately, we cannot do the same for zones (or for sensors if we do zones), since the edges may disallow a zone and a sensor on the same unit.

3 The Case of *InterUnitCap* $= 2$

We now turn to the announced special Partner Units Problem where the number of neighbors of any given unit in a solution is bounded by 2, i.e. *InterUnitCap* $= 2$. This version of the PUP— the SPUP—, which is of high industrial relevance, can be tackled directly. We will do this by giving an algorithm that decides the SPUP in NLOGSPACE by exploiting the notion of a path decomposition of a given graph.

For ease of presentation in the sequel we make the simplifying assumption that the underlying bipartite graph is connected. This does not affect solutions of the SPUDP and the SPUSP, where the connected components can be tackled independently. For optimal solutions, however, the connected components of an underlying graph will have to be considered simultaneously; cf. the discussion in section 3.4.

3.1 Path Decompositions

As already stated our algorithm exploits the notion of a path decomposition of a certain width. We next recall the respective definitions; we assume the reader is familiar with the basic notions from graph theory.

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

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

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

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

Condition (3) 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 $\mathrm{pw}(G)$ of a graph is the minimum width over all its path decompositions.

3.2 Basic Properties of the SPUP

We proceed by identifying basic properties of the SPUP. The key observation 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; 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 zones-and-sensors-graph G:

Theorem 3.1 (SPUP is Path-Decomposable)**.** *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 $\leq (3 * 2 * \textit{UnitCap}) - 1$, with the following special properties:*

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

(b) 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.

(c) 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).

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

(e) 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 $1 \leq i \leq n - 1$.

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

Proof. If G is solvable then there is a solution G^* whose unit graph is a cycle $U_1, \ldots, U_n, U_1$. Consider the path decomposition $(P = W_1, \ldots, W_{n-1}, \chi)$ where $\chi(w_i) = f^{-1}(U_1) \cup f^{-1}(U_i) \cup f^{-1}(U_{i+1})$. This is indeed a path decomposition:

- Every edge (V_1, V_2) is in some bag. Assume V_1 and V_2 are assigned to two different connected units U_i and U_{i+1}. Then $\{V_1, V_2\} \subseteq \chi(W_i)$.
- The connectedness condition is satisfied: For the vertices connected to unit U_1 the induced subgraph is P. All other vertices occur in at most two consecutive bags.
- Every bag contains $\leq (3 * 2 *$ UnitCap$)$ elements; hence $\text{pw}(G) \leq (3 * 2 *$ UnitCap$) - 1$.

An optimal path decomposition of the complete bipartite graph $K_{n,n}$ with $n = 3 *$ UnitCap has width $(3 * 2 *$ UnitCap$) - 1$; cf. figure 1. Hence the bound is tight. The conditions $(a - f)$ are easily seen to hold for the path decomposition constructed above. □

Intuitively, the vertices in the sets $\mathcal{S}_1$ and $\mathcal{S}_2$ from condition (b) 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 neighbors might only appear on the last unit U_n. If all neighbors 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 $2 * \mathit{UnitCap}$.

3.3 An Algorithm for the SPUP

By Theorem 3.1 we know that if a SPUP instance is solvable then there is a path decomposition with specific properties. But we still need an algorithm for finding such suitable path decompositions. Many algorithms for finding path decompositions of bounded width have been proposed in the literature. But, for the SPUP we want to find 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.

Below we introduce an algorithm that fits the bill; it is inspired by the algorithm for finding hypertree decompositions from [12]. This non-deterministic algorithm does the following: 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 [4].

In addition to the bags the unit function is guessed, too. According to condition (d) from above all neighbors 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.

Due to the different roles played by the units that make up a bag, the DECIDESPUP algorithm operates at the level of units rather than bags.

DECIDESPUP(G)

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($C_R, \langle U_1, U_2\rangle, \langle U_{i-1}, U_i\rangle$)

```
1  if C_R = ∅
2     then
3          if ∀V ∈ U_1 nb(V) ⊆ U_1 ∪ U_2 ∪ U_i and
           ∀V ∈ U_i nb(V) ⊆ U_{i-1} ∪ U_i ∪ U_1
4            then ACCEPT
5            else REJECT
6     else
7          Guess non-empty U_{i+1} ⊆ V(⋃ C_R)
           with |U_{i+1} ∩ V_1| ≤ UnitCap ≥ |U_{i+1} ∩ V_2|
8          For V ∈ U_i check nb(V) ⊆ (U_{i-1} ∪ U_i ∪ U_{i+1})
9          C'_R ← (C_R \ U_{i+1})
10         DECIDESPUP (C'_R, ⟨U_1, U_2⟩, ⟨U_i, U_{i+1}⟩)
```

Upon initialization (l. 1–5) the first two units are guessed as subsets of the vertices of the input G (l. 1). Throughout a run of the algorithm the remaining components are stored in C_R (l. 2). We then proceed to the recursive case (l. 3–5).

While recursively processing G (l. 1–10) we consider two cases: (1) If there are no remaining components (l. 1) we check the termination condition in l. 3. The neighbors of U_1 have to appear somewhere on the first, second, or last unit, while the neighbors of the last unit have to appear somewhere on the second-to-last, last, or first unit. Hence we store U_1 and U_2, as well as a a "predecessor" unit U_{i-1} and a "middle" unit U_i throughout a run of the algorithm; upon termination U_{i-1} is the second-to-last, and U_i is the last unit in the cycle. The recursive procedure is first called with $U_{i-1} = U_1$ and $U_i = U_2$, and at each step the contents of the current bag is given by the union of U_1 with $U_{i-1} \cup U_i$. (2) Otherwise, a "successor" unit U_{i+1} is guessed (l. 7). In a solution, all neighbors of vertices assigned to U_i are guaranteed to appear in $U_{i-1} \cup U_i \cup U_{i+1}$ — this is checked in l. 8. For U_{i-1} this will already have been established (if $i > 2$), and hence in the next step U_i and U_{i+1} together with U_1 are again a proper separator. In l. 9 the vertices just assigned to U_{i+1} are deleted from the remaining components, and in l. 10 the same procedure is called recursively.

Using this algorithm we can show the following:

Theorem 3.2 (Tractability of SPUDP)**.** *The decision problem for the* SPUP *is solvable by the algorithm* DECIDESPUP *in* NLOGSPACE *for InterUnitCap* $= 2$ *and any given fixed value of UnitCap.*

Proof. First observe that if the algorithm accepts then there is a solution for the SPUP; moreover, if there is a solution of the SPUP, this clearly can be guessed. We still have to show that the workspace required by DECIDESPUP is logarithmic in the size of the input. The size of the currently retained units is bounded; hence these can be stored in logarithmic space. Moreover, the currently retained units separate the part of the input graph that has already been processed from the remaining components. Hence we only have to represent the remaining components. At each step their number is bounded by $2 * 2 *$ *UnitCap*: For the current units U_{i-1} and U_i there can be at most $2 *$ *UnitCap* neighbors not yet assigned; in the worst case these can all

belong to different connected components. Each of the remaining connected components can be represented by a single vertex; hence we can get by with logarithmic space. □

3.3.1 Answer Extraction

For actually obtaining a solution to a SPUP instance we face the following problem: In general it is not possible to remember the contents of all the bags in logarithmic space. Theoretically this problem can be solved as follows: On a first accepting run of DECIDESPUP we clearly can remember the first bag's contents in logarithmic space. We can then run DECIDESPUP again with a fixed first bag, and so forth. Hence the following holds:

Theorem 3.3 (Tractability of SPUSP)**.** *The problem of finding a solution to the* SPUP *is solvable in* NLOGSPACE *for InterUnitCap* $= 2$ *and any given fixed value of UnitCap.*

Note that the problem of answer extraction disappears when actually implementing the non-deterministic algorithm on a deterministic computer; cf. Section 3.5.

3.3.2 Towards an Efficient Algorithm

We next make a number of observations that can be exploited to turn DECIDESPUP into a practically efficient algorithm.

Guiding the Guessing Not all zones and sensors assigned to units have to be chosen randomly. At most *UnitCap* neighbors of sensors and zones on the first unit can be assigned to the last unit. Hence the following holds:[2]

$$|\mathrm{nb}_s(U_1) \setminus (U_1 \cup U_2)| \leq \mathit{UnitCap} \geq |\mathrm{nb}_z(U_1) \setminus (U_1 \cup U_2)|.$$

Moreover, the neighbors 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:

$$(\mathrm{nb}_s(U_i) \setminus (U_i \cup U_{i-1})) \subseteq U_{i+1} \supseteq (\mathrm{nb}_z(U_i) \setminus (U_i \cup U_{i-1})).$$

Finding Optimal Solutions First 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 3.4 (Tractability of SPUOP)**.** *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.

[2]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.

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.

3.4 SPUOP and Multiple Connected Components

Next let us discuss the problem of finding optimal solutions when the input graph consists of more than one connected component. Here, 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). A priori it is unclear which of the two choices leads to better results. E.g. 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. We leave the complexity of the SPUOP on arbitrary input graphs as an open problem — but we are able to show the following:

Theorem 3.5 (Tractability of SPUOP on Multiple Connected Components)**.** *For $\mathit{InterUnitCap} = 2$ and any given value of UnitCap the optimization problem for the* SPUP *on multiple connected components is solvable in* NLOGSPACE *if there are only logarithmically many connected components in the input graph.*

Proof. (Sketch) Let a graph with c connected components be given as the union $\bigcup_{i=1..c} G_i$ of bipartite graphs $G_i = (\mathcal{V}_{1,i}, \mathcal{V}_{2,i}, \mathcal{E}_i)$. The possible number of unit graphs in optimal solutions ranges from 1 to c. Set $\mathcal{V}_1 = \bigcup_i \mathcal{V}_{1,i}$ and $\mathcal{V}_2 = \bigcup_i \mathcal{V}_{2,i}$. The number of units used is bounded by $lb = \lceil \frac{\max(|\mathcal{V}_1|, |\mathcal{V}_2|)}{\mathit{UnitCap}} \rceil$ from below and $ub = \sum_i \max(|V_{1,i}|, |V_{2,i}|)$ from above. For the lower bound we assume all components fit on a single unit graph; for the upper bound we consider the c individual upper bounds separately.

Next observe that also in the case of multiple connected components a solution to the SPUP gives rise to a path decomposition of a special form: Assume a SPUP instance given by a graph $G = \bigcup G_i$ is solvable with a solution graph G^* consisting of m connected components, and using n units in total. By concatenating the path decompositions as constructed in the proof of Theorem 3.1, we obtain a single path decomposition of length $n - m$. Mutatis mutandis an appropriate version of Theorem 3.1 is obtained.

Likewise the algorithm DECIDESPUP can be adjusted to guess appropriate path decompositions: We have to distinguish between remaining components that are currently being processed and those that are still untouched. The key is then to ensure that a new "cyclic" segment of the path decomposition may only be started if the set of remaining components that are still currently being processed is empty; i.e. there are no unassigned neighboring sensors or zones left.

Observe that the total number of remaining connected components at each step is now bounded by $(4 * \mathit{UnitCap}) + c$, where c is the number of connected components in the input graph. Hence we stay within the logarithmic space bound if there are not more than logarithmically many connected components in the input graph. □

3.5 Implementation

We prototypically implemented the DECIDESPUP algorithm in Java, replacing the non-determinism by a backtracking search mechanism. Our implementation can only handle connected input graphs.

In [13] a deterministic backtracking version of the non-deterministic hypertree decomposition algorithm from [12] is described, and the issues we face when making DECIDESPUP deterministic are very similar. To avoid repeated sub-computations we store pairs of bags and remaining components that could not be decomposed:

Observation 2 (Avoidable Sub-Computations)**.** *Assume a pair of a bag $\mathcal{B}$ and a set of remaining components C_R could not be decomposed by* DECIDESPUP*. If the same pair $\langle \mathcal{B}, C_R \rangle$ occurs again on a run of* DECIDESPUP *it also cannot be decomposed.*

If we use iterative deepening search we also have to store the number of remaining units when encountering a dead-end — it may be possible to decompose the remaining components using more units. We don't store successful pairs — the first such pair occurs when finding a solution.

It turns out that for identifying unsuccessful pairs of bags and remaining components it is enough to store the bag plus the unassigned neighbors:

Lemma 3.6 (Identifying Remaining Components)**.** *Given the contents of a bag and the set of currently unassigned neighbors at any step throughout a run of* DECIDESPUP *the remaining components are uniquely determined.*

Proof. Assume to the contrary that we have calls

1) DECIDESPUP($C_R, \langle U_1, U_2 \rangle, \langle U_{i-1}, U_i \rangle$, and

2) DECIDESPUP($C'_R, \langle U_1, U_2 \rangle, \langle U_{i-1}, U_i \rangle$,

both with the same unassigned neighbors $\mathcal{V}$ (of U_1 and U_i).

First assume that we have different remaining connected components C_R and C'_R on the same set of vertices. This immediately leads to a contradiction.

Next assume that there is a vertex $V_0 \in C_R$ that is not in C'_R, and hence not in $\mathcal{V}$. This V_0 cannot be part of the current bag $\mathcal{B} = U_1 \cup U_{i-1} \cup U_i$ as this is the same in the calls 1) and 2). Hence assume that, in the run leading to the call 2), V_0 is assigned to U_j where $j < (i-1)$. We know that there is a path $V_0, V_1, \ldots, V_n$ in C_R leading from V_0 to some $V_n \in \mathcal{V}$ such that none of the $V_i, 0 \leq i \leq n$ is in $\mathcal{B}$. Hence, in the run leading to the call 2), for one of the pairs V_i, V_{i+1} the test $\text{nb}(V_i) \subseteq (U_{k-1} \cup U_k \cup U_{k+1})$ $(k < i)$ must have failed.

□

As there are only polynomially many possible pairs of current bags and unassigned neighbors, and each of them is constructed at most once, the overall runtime of our implementation of the DECIDESPUP algorithm is polynomial, too.

In order to detect no-good branches in the search tree early we implemented a form of two-step forward-checking: We check whether there is enough space for the open neighbours of the current unit on the current plus the next unit (step one), and do the same for the open neighbours of these open neighbours (step two).

Finally observe that for the backtracking search we have to store the choices made, and hence answer extraction is easy.

4 Standard Models for the General Case

We are next going to outline encodings of the PUP where *InterUnitCap* is an arbitrary fixed constant. Due to cost considerations we are especially interested in the optimization version of the PUP: We want to minimize the number of expensive units used, but do not consider the cost for the cheap connections between them.

In particular, we show how the problem can be encoded in the frameworks of propositional satisfiability testing (SAT), integer programming (IP), and constraint solving (CSP), all of which can be considered as state-of-the-art for optimization problems [14]. In addition we will also describe an encoding in answer set programming (ASP), a currently very successful knowledge representation formalism.

4.1 Answer Set Programming

First, we show how to encode the PUP in answer set programming [10, 15] which has its roots in logic programming and deductive databases. This knowledge representation language is based on a decidable fragment of first-order logic and is extended with language constructs such as aggregation and weight constraints. Already the propositional variant allows the formulation of problems beyond the first level of the polynomial hierarchy. In case standard propositional logic is employed [3], an answer set corresponds to a minimal logical model by definition of [15].

In our encodings a solution (i.e. a configuration) is the restriction of an answer set to those literals that satisfy the defined solution schema.

To encode a PUP instance in ASP we represent the zones and sensors by the unary predicates `zone/1` and `sensor/1`. The edges between zones and sensors are represented by the binary predicate `zone2sensor/2`. The number of available units `lower` $= \left\lceil \frac{\max(|Sensors|,|Zones|)}{UnitCap} \right\rceil$, `unitCap` and `interUnitCap` are each specified by a constant. The PUP is then encoded via the following logical sentences employing the syntax described in [2]:

```
(1)  unit(1..lower).
(2)  1 { unit2zone(U,Z) : unit(U) } 1 :- zone(Z).
(3)  1 { unit2sensor(U,S) : unit(U) } 1 :- sensor(S).
(4)  :- unit(U), unitCap+1 { unit2zone(U,Z): zone(Z) }.
(5)  :- unit(U), unitCap+1 { unit2sensor(U,S): sensor(S) }.
(6)  partnerunits(U,P) :- unit2zone(U,Z), zone2sensor(Z,S),
                          unit2sensor(P,S), U!=P.
(7)  partnerunits(U,P) :- partnerunits(P,U), unit(U), unit(P).
(8)  :- unit(U), interUnitCap+1 { partnerunits(U,P): unit(P) }.
```

The first statement `unit(1). unit(2). ... unit(lower).` generates the required number of units represented as facts. The second and the third clause ensure that each zone and sensor is connected to exactly one unit. The edges between units and zones (rsp. sensors) are expressed by

[3] All literals in rules are negation free. $\bot$, $\rightarrow$, $\wedge$, $\vee$ are used to formulate (disjunctive) rules.

`unit2zone`/2 (rsp. `unit2sensor`/2) predicates. We use cardinality constraints [20] of the form $l\ \{L_1, \ldots, L_n\}\ u$ specifying that at least l but at most u literals of $L_1, \ldots, L_n$ must be true. So called *conditions* (expressed by the symbol ":") restrict the instantiation of variables to those values that satisfy the condition. For example, in the second rule, for any instantiation of variable `Z` a collection of ground literals `unit2zone(U,Z)` is generated where the variable `U` is instantiated to all possible values s.t. `unit(U)` is true. In this collection at least one and at most one literal must be true.

The fourth and the fifth rule guarantee that one unit controls at most *UnitCap* zones and *UnitCap* sensors. In these rules the head of the rule is empty which implies a contradiction in case the body of the rule is fulfilled. The last three rules define the connections between units and limit the number of partner units to *InterUnitCap*. Note that rules 4, 5 and 8 can be rephrased by moving the cardinality constraint on the left-hand-side of the rule and adapting the boundaries. We used the depicted encoding because it follows the Guess/Check/Optimize pattern formulated in [15]. Depending on the particular encoding runtimes may vary.

Alternatively, ASP solvers provide built-in support for optimization by restricting the set of answer sets according to an objective function. For example, for minimizing the number of units, the upper bound on the number of units used has to be provided as a constant `upper` $= \max(|Zones|, |Sensors|)$ for the SPUP, or `upper` $= |Zones| + |Sensors|$ for the PUP. The unit generation rule of the original program (line 1) then has to be replaced by:

```
(1') unit(1..upper).
(2') unitUsed(U):- unit2zone(U,Z).
(3') unitUsed(U):- unit2sensor(U,S).
(4') lower { unitUsed(X):unit(X) } upper.
(5') unitUsed(X):- unit(X), unit(Y), unitUsed(Y), X<Y.
(6') #minimize[unitUsed(X)].
```

Here, the second and the third rule express the property that a used unit always has to be non-empty. Rule `4'` states that the number of used units must be between `lower` and `upper`. Rule `5'` expresses an ordering on the units: units with smaller numbers should be used first. This statement improves the performance of the solver. The last rule expresses that the optimization criterion is the number of units used in a solution.

4.2 Propositional Satisfiability Testing

We next show how to encode the PUP as a propositional satisfiability problem. We are given sensors $[1, S]$, zones $[1, Z]$, and units $[1, U]$, as well as *UnitCap* and *InterUnitCap*.

Let su_{ij} denote that sensor i is assigned to unit j, and zu_{ij} that zone i is assigned to unit j. First of all, every sensor and zone must belong to a unit, so

$$\forall 1 \leq i \leq S \bigvee_{1 \leq j \leq U} su_{ij} \text{ and } \forall 1 \leq i \leq Z \bigvee_{1 \leq j \leq U} zu_{ij}.$$

Furthermore, every sensor and zone belongs to at most one unit, therefore we have

$$\forall 1 \leq i \leq S. \forall 1 \leq j < j' \leq U. \left(\neg su_{ij} \vee \neg su_{ij'}\right)$$

and the same for zones.

Now we need to count both the number of zones and sensors on a unit, and forbid both numbers to be above *UnitCap*. For this we use a sequential counter, similar to the one presented in [21]. Let sc_{ijk} mean that unit j has k sensors assigned (ignore the i for now). We need to say that every sensor counts as one,

$$\forall 1 \leq i \leq S. \forall 1 \leq j \leq U. \left(su_{ij} \rightarrow sc_{ij1}\right),$$

and also that we increment this number when we see something new:

$$\forall 1 \leq i < i' \leq S. \forall 1 \leq j \leq U. \forall 1 \leq k \leq \textit{UnitCap}. \\ \left(su_{i'j} \wedge sc_{ijk} \rightarrow sc_{i'j(k+1)}\right)$$

The fact that we keep track of what we have seen (using index i) is to make sure, for example, that sc_{ij5} is only true if there are five distinct sensors on a unit. Finally, we forbid too many sensors on a unit via

$$\forall 1 \leq i \leq S. \forall 1 \leq j \leq U. \neg sc_{ij(\textit{UnitCap}+1)}.$$

Repeat this trick for zones using zc_{ijk}.

Finally, we need to use the edges. Let sz_{ij} be given, and mean that sensor i has an edge to zone j. Also, let uu_{ij} mean that units i and j are partnered. We need to define this as

$$\forall 1 \leq i \leq S. \forall 1 \leq j \leq Z. \forall 1 \leq k < k' \leq U. \\ \left(((su_{ik} \wedge zu_{jk'}) \vee (su_{ik'} \wedge zu_{jk})) \wedge sz_{ij} \rightarrow uu_{kk'}\right)$$

and also, by symmetry,

$$\forall 1 \leq i < j \leq U. \left(uu_{ij} \rightarrow uu_{ji}\right).$$

Now we can count the partnered units like we did before, using pc_{ijk}, and then forbidding $pc_{ij(y+1)}$. Technically, we don't need both uu_{ij} and uu_{ji}, but having both makes the encoding simpler in the definitions above. We may skip uu_{ii} — but we may also leave them in, as the clauses forcing uu_{ij} have $i < j$, and thus uu_{ii} is never forced. Therefore,

$$\forall 1 \leq i \leq j \leq U. \left(uu_{ij} \rightarrow pc_{ij1}\right),$$

and

$$\forall 1 \leq i < i' \leq U. \forall 1 \leq j \leq U. \left(uu_{i'j} \wedge pc_{ijk} \rightarrow pc_{i'j(k+1)}\right).$$

Finally, we forbid too many partners, and we are done:

$$\forall 1 \leq i \leq j \leq U. \neg pc_{ij(\textit{InterUnitCap}+1)}.$$

4.3 Integer Programming

We next show how the PUP can be encoded into integer programming. If $\textit{InterUnitCap} = 2$ we set $|\text{Units}| = \max(|\text{Sensors}|, |\text{Zones}|)$; otherwise it is $|\text{Units}| = |\text{Sensors}| + |\text{Zones}|$. Then we make matrices of Boolean variables su_{ij} (and zu_{ij}, respectively) sensor s_i (zone z_i) is assigned

to unit u_j. These matrices are constrained to enforce that each sensor/zone is assigned exactly one unit, and that no unit is assigned more than *UnitCap* sensors/zones:

$$\begin{array}{cccc|c} su_{1,1} & su_{2,1} & su_{3,1} & \dots & \sum \leq \mathit{UnitCap} \\ su_{1,2} & su_{2,2} & \dots & \dots & \sum \leq \mathit{UnitCap} \\ su_{1,3} & \dots & \dots & \dots & \sum \leq \mathit{UnitCap} \\ \dots & \dots & \dots & \dots & \dots \\ \hline \sum = 1 & \sum = 1 & \dots & \dots & \end{array}$$

The zone-units matrix looks identical. Next we need a Boolean variable $\mathit{UnitUsed}_i$ that indicates whether u_i is assigned any sensors/zones. This can be achieved by constraints $su_{ji} \leq \mathit{UnitUsed}_i$ and $zu_{ji} \leq \mathit{UnitUsed}_i$, for all j. Observe that in principle even for unused units $\mathit{UnitUsed}_i$ can be set to one — a possibility that will be excluded by the objective function.

For the constraints on the connections between units it is convenient to increase *InterUnitCap* by one, and stipulate that every unit is connected to itself. We then obtain a symmetric matrix of Boolean uu_{ij} variables, which can be used to indicate whether unit i is connected to unit j:

$$\begin{array}{cccc|c} 1 & uu_{1,2} & uu_{1,3} & \dots & \sum \leq \mathit{InterUnitCap} + 1 \\ uu_{2,1} & 1 & \dots & \dots & \sum \leq \mathit{InterUnitCap} + 1 \\ uu_{3,1} & \dots & 1 & \dots & \sum \leq \mathit{InterUnitCap} + 1 \end{array}$$

In addition to enforcing that *InterUnitCap* is not exceeded, the entries in this matrix are subject to the following constraints:

- $uu_{ij} = uu_{ji}$ (symmetry); and
- $uu_{ij} \geq (su_{ki} + zu_{lj}) - 1$, for all connections (s_k, z_l) between sensors and zones — if a sensor s_k and a zone z_l are connected yet assigned different units u_i, u_j then these units are connected.

This model allows more connections between units than are actually needed, in this case mandating a post-processing step for solutions.

As a last constraint we add that the number of units used is bounded from below:

$$\lceil \frac{\max(|\text{Sensors}|, |\text{Zones}|)}{2} \rceil \leq \sum_j \mathit{UnitUsed}_j.$$

Finally, we add the objective function $\sum_j \mathit{UnitUsed}_j$, subject to minimization. As usual, first a linear relaxation with cost C is solved, and only then is the problem solved over the integers, posting the cost C as a lower bound.

4.4 Constraint Satisfaction Problem

Finally, we model the PUP as a CSP by letting sensors and zones be variables $\mathcal{S} = \{s_1, \dots, s_n\}$ and $\mathcal{Z} = \{z_1, \dots, z_m\}$. For the domains we use (a numbering of) the units $U_1, \dots, U_n$.

We post a global cardinality constraint $\texttt{gcc}(U_i, [s_1, \ldots, s_n], c)$ on the sensors for every U_i , where c is a variable with domain $\{0, \ldots, \mathit{UnitCap}\}$, and do likewise for the zones. These constraints ensure that each unit occurs at most *UnitCap* times in any assignment to $\mathcal{S}$ and $\mathcal{Z}$.

Tracking connections between units via Boolean variables is done using a matrix of Boolean uu_{ij} variables as in the integer programming model, but using cardinality constraints to count the number of ones.

In addition for each connection (S, Z) we post implicational constraints that exclude the value j from the domain of sensor s if z is assigned to unit i and $uu_{ij} = 0$ (and vice versa):

$$(s = U_i \wedge uu_{ij} = 0) \rightarrow z \neq U_j \text{ and } (z = U_i \wedge uu_{ij} = 0) \rightarrow s \neq U_j$$

4.5 Adapting the Encodings to *InterUnitCap* $= 2$

To some extent ideas developed in the context of the DECIDESPUP algorithm can be incorporated into the other problem models:

- All models allow symmetry breaking as defined in section 3.3 above, assuming a fixed cyclic layout of the units.
- In the ASP-, the SAT- and the CSP-model we can assume a fixed circular layout of the unit graph because we use iterative deepening search for optimization. We can then post conditional constraints, for each connection between a sensor S and a zone Z, stating that if S is assigned to unit u_i then Z has to be assigned to one of U_{i-1}, U_i, or U_{i+1} (and vice versa). It is not clear how to express this in the integer programming model without sacrificing the objective function.
- In the CSP model we can do something additional: Via a suitable variable ordering we can enforce that a unit for a sensor (or a zone) is chosen only if the sensor (the zone) is connected to an already assigned zone (or sensor). This variable ordering can be computed e.g. by a simple greedy algorithm.Consequently, the number of choices per zone (or sensor) is bounded by three throughout the search, instead of *NoOfUnits*. Both ASP- and SAT-solvers do not usually provide this level of control to the user.

5 Evaluation

We have evaluated our encodings on a set of benchmark instances that we received from our partners in industry. All experiments were conducted on a 3 GHz dual core machine with 4 GB RAM running Fedora Linux, release 13 (Goddard). In general in our experiments we have imposed a ten minute time limit for finding solutions.

For the evaluation of the different encodings of the PUP we use the SAT-solver MINISAT v2.0 [17], the constraint logic programming language ECL^{i}PS^e-Prolog v6.0 [7], and CLINGO v3.0 [2] from the Potsdam Answer Set Solving Collection (Potassco). For evaluating the integer programming model we have used CBC v2.6.2 in combination with CLP v1.13.2 from the COIN-OR project [3], and IBM's CPLEX v12.1 [5].

In the ASP, SAT and CSP models, as well as in DECIDESPUP, we use iterative deepening search for finding optimal solutions, as this has proven to be the most efficient. We did not try this in the integer programming model, as we would lose the objective function in doing so.

The reader is advised to digest the results presented below with caution: We are using both the SAT and the integer programming solvers out of the box, whereas for the CSP model we employ the variable ordering heuristics outlined in the previous section. Moreover, if $\mathit{InterUnitCap} > 2$, for the ASP model we employ the following advanced feature: a portfolio solver CLASPFOLIO, which is a part of Potassco [2], comes with a machine learning algorithm (support vector machine) that has been trained on a large set of ASP programs. CLASPFOLIO analyzes a new ASP program (in our case the PUP program), and configures CLINGO to run with options that have already proved successful on similar programs. It is likely that such machine learning techniques could also be developed and fruitfully applied in the other frameworks.

5.1 Experimental Results

$\mathit{InterUnitCap} = 2$ All instances are based on rectangular floor plans, and all instance graphs are connected. In all instances there is one zone per room, and by default there are sensors on all doors. Only the tri-* instances feature external doors. For an illustration see Figure 2, which shows a rectangular 8×3 floor plan with external doors on two sides of the building.

Apart from that, the instances are structured as follows:

- dbl-* consist of two rows of rooms with all interior doors equipped with a sensor.
- dblv-* are the same, only that there are additional zones that cover the columns.
- tri-* are grids with only some of the doors equipped with sensors. There are additional zones that cover multiple rooms.

The runtimes we obtained for the various problem encodings described above are shown in seconds in Table 1 (a "*" indicates a timeout). The Cost column contains the number of units in an optimal solution; a slash "/" in that column indicates that no solution exists.

$\mathit{InterUnitCap} > 2$ For the general case we have also tested our encodings on a set of benchmark instances where $\mathit{InterUnitCap} = 4$ that we obtained from our partners in industry:

- tri-* are exactly as before, only with $\mathit{InterUnitCap} = 4$.
- grid-* are not full grids, but some doors are missing, and there are no rooms (zones) without doors.

5.2 Analysis

Any conclusions drawn from our experimental results have to be qualified by the remark that, of course, in every solution framework there are many different problem models, and there is no guarantee that our problem models are the best ones possible.

Table 1: Structured Problems with *InterUnitCap* = *UnitCap* = 2

Name	$\|S\|$	$\|Z\|$	Edges	Cost	CSP	SAT	DECIDESPUP	ASP	CBC	CPLEX
dbl-20	28	20	56	14	0.02	0.48	0.01	0.16	14.12	1.53
dbl-40	58	40	116	29	0.28	2.36	0.05	3.93	224.14	13.58
dbl-60	88	60	176	44	0.42	29.74	0.08	*	*	213.58
dbl-80	118	80	236	59	1.14	*	0.16	*	*	522.50
dbl-100	148	100	296	74	1.89	*	0.41	*	*	*
dbl-120	178	120	356	89	3.21	*	0.39	*	*	*
dbl-140	208	140	416	104	5.01	*	0.59	*	*	*
dbl-160	238	160	476	119	13.94	*	0.71	*	*	*
dbl-180	268	180	536	134	20.07	*	0.87	*	*	*
dbl-200	298	200	596	149	14.4	*	1.08	*	*	*
dblv-30	28	30	92	15	0.09	0.42	65.49	0.26	37.18	2.93
dblv-60	58	60	192	30	0.26	3.15	*	1.94	*	*
dblv-90	88	90	292	45	0.82	12.54	*	27.35	*	*
dblv-120	118	120	392	60	1.85	41.65	*	13.92	*	*
dblv-150	148	150	492	75	3.48	20.97	*	29.54	*	*
dblv-180	178	180	592	90	6.20	44.28	*	54.50	*	*
tri-30	40	30	78	20	1.07	0.79	0.50	0.41	45.17	78.75
tri-32	40	32	85	20	0.64	0.74	*	0.26	55.20	4.66
tri-34	40	34	93	/	21.10	22.77	*	0.89	74.78	5.06
tri-60	79	60	156	40	158.49	315.42	114.08	4.40	*	108.01
tri-64	79	64	170	/	*	379.36	*	43.88	*	76.26

Let us begin our analysis of the results by highlighting a peculiarity of the PUP: While it is possible to construct instances that require more than the minimum number of units, it is not straight-forward to do so, and such instances also appear to be rare in practice: In our experiments in no solution are there more units than the bare minimum required. It is clear that iterative deepening search thrives on this fact, whereas the integer programming model suffers.

InterUnitCap = 2 The combination of assuming a fixed cyclic unit graph together with iterative deepening search resulted in drastic speedups for the ASP, SAT, and CSP solvers. Symmetry breaking did not have much effect — except on the unsolvable instances.

The ASP and the SAT encoding show broadly similar behavior: Both CLINGO and MINISAT use variations of the DPLL-procedure [6] for reasoning. Oddly, they even both get faster at some point as problem size increases on the dblv-* instances. However, CLINGO does significantly better on the grid-like instances. Interestingly, machine learning did not help for the ASP encoding specialized to *InterUnitCap* = 2; hence the results shown were obtained using both solvers out of the box.

For the CSP encoding the variable ordering is the key to the good results: Without the variable ordering the CSP model performs quite poorly. The absence of a similar variable selection

Table 2: Structured Problems with *InterUnitCap* $= 4$ and *UnitCap* $= 2$

Name	$\|S\|$	$\|Z\|$	Edges	Cost	CSP	SAT	ASP	CBC	CPLEX
tri-30	40	30	78	20	0.12	2.40	0.40	182.91	24.79
tri-32	40	32	85	20	0.14	1.91	0.66	270.27	20.84
tri-34	40	34	93	20	*	1.98	0.60	331.29	*
tri-60	79	60	156	40	0.52	*	11.07	*	*
tri-64	79	64	170	40	*	*	7.61	*	*
tri-90	118	90	234	59	1.50	401.44	332.34	*	*
tri-120	157	120	312	79	3.37	*	*	*	*
grid-1	100	79	194	50	*	78.19	31.45	*	*
grid-2	100	77	194	50	*	90.89	18.91	*	*
grid-3	100	78	194	50	*	88.87	25.72	*	*
grid-4	100	80	194	50	*	95.12	24.66	*	*
grid-5	100	76	194	50	*	454.42	48.88	*	*
grid-6	100	78	194	50	*	204.85	9.15	*	*
grid-7	100	79	194	50	*	112.36	12.89	*	*
grid-8	100	78	194	50	*	*	11.89	*	*
grid-9	100	76	194	50	*	91.62	19.71	*	*
grid-10	100	80	194	50	*	545.16	13.54	*	*

mechanism from both ASP and SAT in our experiments might explain the surprising superiority of CSP on most benchmarks.

The inconsistent results for DECIDESPUP are particularly striking. On the one hand, it performs excellently on the dbl-* instances. But in general, it disappoints. Possibly this might be due to the following: DECIDESPUP has a "local" perspective on the problem, that is, it only can see the current and past units; the subsequent units are only created at runtime. In all the other encodings all units are present from the beginning, something which, in one way or another, facilitates propagating the current variable assignment to other units.

The IP encoding is not yet fully competitive. It particularly struggles with the dblv-* instances. In general, the commercial CPLEX is at least one order of magnitude faster than the open source CBC.

It is also interesting to compare the dbl-* with the dblv-* instances, as the latter are obtained from the former by adding constraints. Both CLINGO and MINISAT thrive on the additional constraints, contrary to ECL^{i}PS^e, CBC, CPLEX and DECIDESPUP.

InterUnitCap > 2 In this setting, for finding solutions the symmetry breaking methods from section 2.6 did increase computation time for the CSP, the SAT, and the IP model. However, symmetry breaking again does help when proving an instance unsatisfiable. The results in Table 2 were obtained without symmetry breaking.

If CLASPFOLIO's machine learning database is not used to configure options of CLINGO, then the two DPLL-based programs again perform quite similar, with Clingo slightly having the edge (results not shown). With machine learning CLINGO clearly is the winner, with the main benefits

stemming from the following: Use the VSIDS heuristics [18] instead of the BerkMin heuristics [11], and exploit local restarts [19]. Note that MINISAT also uses the VSIDS heuristics.

Interestingly, the CSP-encoding now disappoints. Given that the same variable ordering is used, this may have to be attributed to insufficient propagation when tracking the connections between units.

Again our IP encoding is not on par yet. But for this encoding comparing the instances tri-30,32,34 in Tables 1 and 2 is particularly instructive: This is basically the same model in both settings, only that in the latter case there are more variables due to the higher upper bound on the number of required units.

6 Conclusion

In this work we have advanced the state-of-the-art of the Partner Units Problem, a new hard configuration problem. We have presented and evaluated encodings of this problem in the frameworks of answer set programming, constraint solving, integer programming, and satisfiability testing. We have also evaluated our prototypical implementation of a dedicated algorithm for the PUP that has recently allowed us to prove the tractability of one class of problem instances that is of great interest for our partners in industry.

Interestingly, integer programming methods are of no avail for this problem in general. Also, our prototype implementation of the dedicated algorithm did not perform to our satisfaction. For the class of problem instances where the number of partner units is limited to two the encoding of the problem as a CSP in ECL^{i}PS^e-Prolog proved to be practically the most efficient. For the general case the encoding as an ASP in CLINGO is the best that we have.

There is still significant work to be done on the PUP: Almost all interesting complexity questions are still open. We believe that a thorough investigation of these questions is the prerequisite for eventually finding better algorithms for the general case of the PUP.

Acknowledgment. Work in Klagenfurt funded by FFG FIT-IT SemSys 825071. Work in Oxford funded by EPSRC Grant EP/G055114/1 "Constraint Satisfaction for Configuration: Logical Fundamentals, Algorithms and Complexity". G. Gottlob's work, in particular, with respect to specific topics arising in the context of the co-operation with Siemens Austria and the University of Klagenfurt, was also partially funded by the FFG FIT-IT project RECONCILE. G. Gottlob would also like to acknowledge the Royal Society Wolfson Research Merit Award.

References

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

[2] The Potsdam Answer Set Solving Collection. `http://potassco.sourceforge.net/`.

[3] COIN-OR CLP/CBC IP solver. `http://www.coin-or.org/`.

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

[5] IBM ILOG CPLEX IP solver. `http://www.ibm.com/`.

[6] Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3), 1960.

[7] ECL^{i}PS^e-Prolog. `http://eclipseclp.org/`.

[8] 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.

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

[10] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP'88*, 1988.

[11] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Proceedings of DATE'02*, 2002.

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

[13] G. Gottlob and M. Samer. A backtracking-based algorithm for hypertree decomposition. *ACM Journal of Experimental Algorithmics*, 13, 2008.

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

[15] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3), 2006.

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

[17] Minisat SAT solver. `http://www.minisat.se`.

[18] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of DAC'01*, 2001.

[19] Vadim Ryvchin and Ofer Strichman. Local restarts. In *Proceedings of SAT'08*, 2008.

[20] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2), 2002.

[21] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming - CP 2005*. Springer, 2005.