Tackling the Partner Units Configuration Problem*

Markus Aschinger, Conrad Drescher, Georg Gottlob, Peter Jeavons, Evgenij Thorstensen
Computing Laboratory
University of Oxford
firstname.lastname@comlab.ox.ac.uk

Abstract

The Partner Units Problem is a specific type of configuration problem with important applications in the area of surveillance and security. In this work we show that a special case of the problem, that is of great interest to our partners in industry, can directly be tackled via a structural problem decompostion method. Combining these theoretical insights with general purpose AI techniques such as constraint satisfaction and SAT solving proves to be particularly effective in practice.

1 Introduction

In this paper we address the *Partner Units Problem (*PUP*)* [Falkner *et al.*, 2010], a configuration problem of high relevance to surveillance and security applications where a large number of sensors is divided into possibly overlapping zones. More specifically, the PUP deals with the configuration of a network of control units ("units") for the sensors and zones.

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. These control units can be connected to a given 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 must be kept simple because of capacity constraints such as in intelligent traffic management or safeguarding semi-automated transport systems. The PUP is used as a benchmark instance at this year's answer set programming competition [ASPcomp, 2011].

Figure 1 shows a PUP instance and a solution for the case $UnitCap = InterUnitCap = 2$: 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.

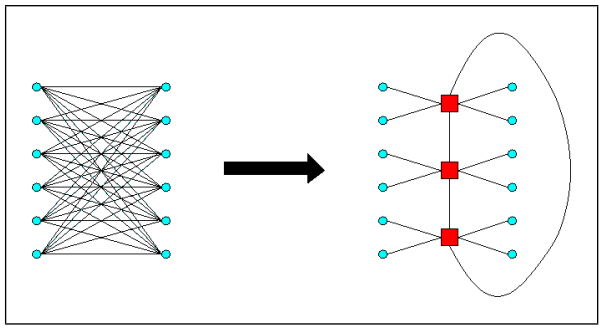

Figure 1: Solving a $K_{6,6}$ Partner Units Instance — Partitioning Sensors and Zones into Units

Let us emphasize that the PUP is a typical configuration problem in the sense of [Mittal and Frayman, 1989]: Connect a fixed finite set of given components so as to meet a given objective while respecting given constraints. Solving such configuration problems is one of the major success stories of applied AI research: Starting from early work on rule based systems [McDermott, 1982], manifold general purpose AI techniques such as CSP and SAT solving, heuristic search, and description logics have successfully been applied to configuration — for a recent survey see [Junker, 2006].

Due to cost considerations our partners in industry are primarily interested in finding optimal solutions, that is, solutions that use a minimum number of units. The rationale be-

*Work funded by EPSRC Grant EP/G055114/1. Cooperation with Siemens Austria partially funded by FFG FIT-IT Grant 825071.

[1] For ease of presentation we assume that *UnitCap* is the same for zones and sensors.

hind this optimization criterion is that (a) units are expensive, and (b) costs for connections are negligible.

Formally, the following are the three most important reasoning tasks for the PUP:

- Decide whether there is a solution (PUDP).
- Find a solution (PUSP).
- Find an optimal solution; i.e. one that uses a minimum number of control units (PUOP): We treat this as deciding whether there is a solution that uses n or fewer units.

In this paper we study the special version of the PUP where each unit can be connected to at most k sensors or zones (*UnitCap* $= k$) and to at most two other units (*InterUnitCap* $= 2$). This version of the problem (the SPUP) is precisely the problem-description we got from our industrial project partner.

We first tried to tackle the SPUP via CSP and SAT solving technology and answer set programming, and previous approaches additionally include heuristic search methods such as simulated annealing — none of these approaches were much use on the instances encountered in practice. The insight that these instances could be directly tackled via the algorithm presented in this paper constituted a major breakthrough performance-wise. A second major breakthrough occurred when incorporating these theoretical insights into the aforementioned general AI techniques.

A Guide to the Paper. The remainder of this paper is organized as follows: In Section 2 we formally introduce the problem and state general problem properties. Among other things we show that in its most general form the PUP is NP-hard (Theorem 1). Then in Section 3 we show that the SPUDP and the SPUSP are tractable. We do this by giving the NLOGSPACE algorithm DECIDESPUP that solves the problem by constructing a path decomposition with specific properties. We also show that the SPUOP is tractable if there are not more than logarithmically many connected components in the instance. In the same section we discuss our implementation of (a deterministic version of) the DECIDESPUP algorithm. In Section 4 we briefly evaluate the performance of our algorithm and an adapted CSP encoding. In Section 5 we conclude the paper with notes on related work and an outlook on future research.

We have also been working on encodings of the general version of the PUP— that is where both *UnitCap* and *InterUnitCap* are arbitrary fixed constants — in the frameworks of answer set, integer, and constraint programming as well as SAT solving. These encodings together with a more thorough empirical evaluation of the DECIDESPUP algorithm are presented in [Aschinger *et al.*, 2011].

2 Formal Definition of the Partner Units Problem and Basic Facts

Formally, the PUP consists of partitioning the vertices of a bipartite graph $G = (V_1, V_2, E)$ into a set $\mathcal{U}$ of units such that each unit

- contains at most *UnitCap* vertices from V_1 and at most *UnitCap* vertices from V_2; and
- has at most *InterUnitCap* adjacent units, where the units U_1 and U_2 are adjacent whenever $v_i \in U_1$ and $v_j \in U_2$ and $(v_i, v_j) \in E$.

To every solution of the PUP we can associate a *solution graph*. For this we associate to every unit $U_i \in \mathcal{U}$ a vertex v_{U_i}. Then the solution graph G^* has the vertex set $V_1 \cup V_2 \cup \{v_{U_i} \mid U_i \in \mathcal{U}\}$ 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*.

Unfortunately, in its most general form the PUOP is intractable:

Theorem 1 (PUOP is Intractable)**.** *The* PUOP *is* NP*-complete when InterUnitCap* $= 0$*, and UnitCap is part of the input.*

Proof. Membership in NP is obvious. For hardness, we reduce from BINPACKING, given by natural numbers $i_1, \ldots, i_n$, *BinSize* and k. We make a PUOP instance by creating for every i_j a biclique between one fresh sensor and i_j fresh zones, setting *UnitCap* $=$ *BinSize* and *InterUnitCap* $= 0$. A packing with k or 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 [Garey and Johnson, 1979]. □

Observe, however, that the above theorem concerns the case of optimally configuring one parameter of the units, namely *UnitCap*, rather than finding a configuration that respects the given unit properties.

On the other hand it is not hard to show the following:

Lemma 1 (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* U_i adjacent to 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 the U_i. □

Let us next point out that the number of units used when solving an instance $G = (V_1, V_2, E)$ is bounded from below by $lb = \lceil \frac{\max(|V_1|,|V_2|)}{\textit{UnitCap}} \rceil$. Clearly it can also be bounded from above by $ub = |V_1| + |V_2|$ — we never need empty units. If *InterUnitCap* $= 2$ and *UnitCap* > 1 we can show that the stronger $ub = \max(|V_1|, |V_2|)$ holds for connected instances. We conjecture that this also holds for *InterUnitCap* > 2, but have so far been unable to prove it. Now, if there are multiple connected components C_i in the instance with upper bounds ub_i, then we have $ub = \sum ub_i$.

3 The Case of *InterUnitCap* $= 2$

We now turn to the SPUP, 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* $= 2$. We will directly tackle the SPUP by giving an algorithm that decides the SPUDP in NLOGSPACE by computing a special path decomposition of the instance 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

Let us next formally introduce path decompositions. A path decomposition of a graph $G = (V, E)$ is a pair $\mathcal{P} = (P, \chi)$ such that P is a simple path (W, F) — i.e., P does not contain cycles. The function χ associates to every $w \in W$ a subset $B \subseteq V$ such that

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

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

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

Condition (3) is called the connectedness condition. The subsets B associated with the path vertices are called bags. The width of a path decomposition is $\max_{w \in W}(|\chi(w) - 1|)$. 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.

Now every solution of a SPUP instance given by a zones-and-sensors-graph G gives rise to a path decomposition with special properties:

Theorem 2 (SPUP is Path-Decomposable)**.** *Assume a* SPUP *instance given by a graph $G = (V_1, V_2, E)$ is solvable using n units, that is $|\mathcal{U}| = n$. Let f be the unit function that associates vertices from G to $\mathcal{U}$. Then there is a path decomposition $\mathcal{P} = (P, \chi)$ of G of pathwidth $\leq (3{*}2{*}$UnitCap$)-1$:*

Proof. If G is solvable then there is a solution graph G^* whose unit graph is a cycle $v_{U_1}, \ldots, v_{U_n}, v_{U_1}$. Consider $\mathcal{P} = (P = w_1, \ldots, w_{n-1}, \chi)$ where $\chi(w_i) = f^{-1}(U_1) \cup f^{-1}(U_i) \cup f^{-1}(U_{i+1})$. This $\mathcal{P}$ 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 in $\mathcal{P}$ contains $\leq (3 * 2 *$ UnitCap$)$ elements; hence $\mathrm{pw}(\mathcal{P}) \leq (3 * 2 *$ 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 following conditions are easily seen to hold for the path decomposition $\mathcal{P}$ constructed above:

(a) The length of P is $n - 1$; $P = w_1, \ldots w_{n-1}$.

(b) There are sets $S_1 \subseteq V_1$, $S_2 \subseteq V_2$ with $|S_i| \leq$ UnitCap such that $S_1 \cup S_2$ are in every bag of $\mathcal{P}$.

(c) Apart from $S_1 \cup S_2$ each bag contains at most $2{*}$UnitCap elements from V_1 (or V_2, respectively).

(d) For any vertex $v \in V_1 \cup V_2$ all neighbors of v appear in three consecutive bags of $\mathcal{P}$ (assuming the first and last bag to be connec ted).

(e) For each bag $\chi(w_i)$ of $\mathcal{P}$ 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) $S_1 = f^{-1}(U_1) \cap V_1$ and $S_2 = f^{-1}(U_1) \cap V_2$.

Intuitively, the vertices in the sets S_1 and 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 S_1 and 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.

3.3 An Algorithm for the SPUP

By Theorem 2 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 $\mathcal{P}$ 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 [Gottlob *et al.*, 2002]. 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 [Cook, 1971].

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 $\mathcal{P}$. 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 $U_1, U_2 \subseteq V(G)$ with $|U_i \cap V_1| \leq \mathit{UnitCap} \geq |U_i \cap V_2|$
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** $C_R = \emptyset$
2 **then**
3 **if** $\forall v \in U_1$ $\mathrm{nb}(v) \subseteq U_1 \cup U_2 \cup U_i$ and $\forall v \in U_i$ $\mathrm{nb}(v) \subseteq U_{i-1} \cup U_i \cup U_1$
4 **then ACCEPT**
5 **else REJECT**
6 **else**
7 Guess non-empty $U_{i+1} \subseteq V(\bigcup C_R)$ with $|U_{i+1} \cap V_1| \leq \mathit{UnitCap} \geq |U_{i+1} \cap V_2|$
8 For $v \in U_i$ check $\mathrm{nb}(v) \subseteq (U_{i-1} \cup U_i \cup U_{i+1})$
9 $C'_R \leftarrow (C_R \setminus U_{i+1})$
10 DECIDESPUP ($C'_R, \langle U_1, U_2\rangle, \langle U_i, U_{i+1}\rangle$)

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 (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 $2 * 2 * \mathit{UnitCap}$: For the current units U_{i-1} and U_i there can be at most $2 * \mathit{UnitCap}$ 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. □

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 4 (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.

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 $G = (V_1, V_2, E)$ 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:

[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.

Corollary 1 (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.

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 *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 5 (Tractability of SPUOP on Multiple Connected Components)**.** *For 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 $G_i = (V_{1,i}, V_{2,i}, E_i)$. The possible number of unit graphs in optimal solutions ranges from 1 to c. Set $V_1 = \bigcup_i V_{1,i}$ and $V_2 = \bigcup_i V_{2,i}$. The number of units used is bounded by $lb = \lceil \frac{\max(|V_1|,|V_2|)}{\mathit{UnitCap}} \rceil$ 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 2, we obtain a single path decomposition of length $n-m$. Mutatis mutandis an appropriate version of Theorem 2 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 [Gottlob and Samer, 2008] a deterministic backtracking version of the non-deterministic hypertree decomposition algorithm from [Gottlob *et al.*, 2002] 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 1 (Avoidable Sub-Computations)**.** *Assume a pair of a bag B and a set of remaining components C_R could not be decomposed by* DECIDESPUP*. If the same pair $\langle B, C_R \rangle$ 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 2 (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 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 V. This v_0 cannot be part of the current bag $B = U_1 \cup U_{i-1} \cup U_i$ 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 $v_n \in V$ such that none of the $v_i, 0 \leq i \leq n$ is in B. Hence, in the run leading to the call 2), for one of the pairs v_i, v_{i+1} the test $\mathrm{nb}(v_i) \subseteq (U_{k-1} \cup U_k \cup U_{k+1})$ $(k < i)$ 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 Evaluation

As already stated in the introduction, in addition to the methods previously used in industry we have also developed encodings of the PUOP in SAT and CSP solving, as well as answer set and integer programming. We then proceeded to adapt these encodings to the SPUOP, combining the key insights from the DECIDESPUP algorithm with advanced solving techniques from the respective areas; these encodings, together with an evaluation of their performance, are described in [Aschinger *et al.*, 2011].

We have used these encodings, both adapted and not, to compare our implementation of the DECIDESPUP algorithm against; the evaluation was done using a set of instances that we received from our partners in industry. A small excerpt from our experimental results is shown in Table 1, where each line is a representative result for a different class of problem instances. We imposed a ten minute time limit, a "*" indicates a timeout, and runtimes are in seconds. The number of units needed in optimal solutions is denoted by "Cost". CSP denotes a PUOP encoding in ECL^{i}PS^e Prolog, and CSP$^{\text{DECIDESPUP}}$ the adaption thereof to the SPUOP.

Table 1: Experimental Results

Edges	Cost	CSP	DECIDESPUP	CSP$^{\text{DECIDESPUP}}$
92	15	*	65.49	0.09
97	33	*	202.48	0.03
156	40	*	114.08	158.49
236	59	*	0.16	1.14

It can be seen that our prototype implementation of DECIDESPUP can find optimal solutions much faster than the (non-adapted) encoding as a CSP. But encodings of the SPUOP in general purpose AI frameworks such as SAT or CSP in general perform much better than our prototype implementation if they are adapted to exploit the key insights of the DECIDESPUP algorithm [Aschinger *et al.*, 2011].

On the one hand, none of the adapted encodings provably runs in polynomial time; on the other hand, these encodings seem to be much better at detecting no-good branches in the search tree early than our implementation that tries to do so using the two-step look-ahead scheme described above.

5 Related and Future Work

In this work we have shown how the Special Partner Units Configuration Problem can directly be tackled by computing a path decomposition of the input graph. More typically, structural problem decomposition methods such as path, tree, and hypertree decompositions are used to identify classes of problem instances that can be solved in polynomial time; for an example of this usage in a logic-based formalism for configuration problems see [Gottlob *et al.*, 2007].

While our results clearly advance the state-of-the art of the PUP, there are a number of questions that deserve further research efforts. One important direction is the study of search heuristics so as to achieve a further speedup of the runtime. On the more theoretical side, it may be worthwhile studying two problems of currently unknown complexity: (i) the PUP and PUOP in case *InterUnitCap* is fixed but greater than 2, and (ii) the case of SPUOP where there are more than $O(\log n)$ connected components in the input graph.

Acknowledgments We greatly acknowledge the constructive criticism we received from the reviewers.

References

[Aschinger *et al.*, 2011] M. Aschinger, C. Drescher, G. Friedrich, G. Gottlob, P. Jeavons, A. Ryabokon, and E. Thorstensen. Optimization Methods for the Partner Units Problem. In *Proceedings of CPAIOR'11*, 2011.

[ASPcomp, 2011] Third ASP Competition. `https://www.mat.unical.it/aspcomp2011/`, 2011.

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

[Falkner *et al.*, 2010] A. Falkner, A. Haselböck, and G. Schenner. Modeling Technical Product Configuration Problems. In *Proceedings of the Configuration Workshop at ECAI'10*, 2010.

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

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

[Gottlob *et al.*, 2002] G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decomposition and Tractable Queries. *Journal of Computer and System Sciences*, 64(3), 2002.

[Gottlob *et al.*, 2007] G. Gottlob, G. Greco, and T. Mancini. Conditional constraint satisfaction: Logical foundations and complexity. In *Proceedings of IJCAI'07*, 2007.

[Junker, 2006] U. Junker. Configuration. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2006.

[McDermott, 1982] J. McDermott. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19, 1982.

[Mittal and Frayman, 1989] S. Mittal and F. Frayman. Towards a generic model of configuraton tasks. In *Proceedings of IJCAI'89*, 1989.