The Computational Complexity of the Minimum Weight Processor Assignment Problem

H.J. Broersma[1], D. Paulusma[2], G.J.M. Smit[2],
F. Vlaardingerbroek[2] and G.J. Woeginger[3]

[1] Department of Computer Science,
University of Durham, Science Labs, South Road, Durham DH1 3LE, England.
hajo.broersma@durham.ac.uk
[2] Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente, 7500 AE Enschede, The Netherlands.
d.paulusma@math.utwente.nl, smit@cs.utwente.nl,
f.vlaardingerbroek@student.utwente.nl
[3] Department of Mathematics and Computer Science
Eindhoven University of Technology, P.O. Box 513,
5600 MB Eindhoven, The Netherlands.
gwoegi@win.tue.nl

Abstract. In portable multimedia systems a number of communicating tasks has to be performed on a set of heterogeneous processors in an energy-efficient way. We model this problem as a graph optimization problem, which we call the minimum weight processor assignment problem. We show that our setting generalizes several problems known in literature, including minimum multiway cut, graph k-colorability, and minimum (generalized) vertex covering. We show that the minimum weight processor assignment problem is NP-hard, even when restricted to instances where the (process) graph is a bipartite graph with maximum degree at most 3, or with only two processors, or with arbitrarily small weight differences, or with only two different edge weights. For graphs with maximum degree at most 2 (or in fact the larger class of degree-2-contractible graphs) we give a polynomial time algorithm. Finally we generalize this algorithm into an exact (but not efficient) algorithm for general graphs.

Keywords: computational complexity, processor assignment.

2000 Mathematics Subject Classification: 05C85, 03D15.

1 Introduction

In portable multimedia systems a number of communicating tasks has to be performed on a set of heterogeneous processors. The key issue in the design of portable multimedia systems is to find a good balance between flexibility and high-processing power on one side, and area and energy-efficiency of the implementation on the other side (cf. [7]).

In this paper we will model a particular problem from this area as a graph optimization problem, in which the vertices of the graph represent the tasks,

and its edges represent the communication between two tasks. Each task must be performed on exactly one processor. Each possible choice of processors involves a certain amount of costs reflecting the energy consumption involved in running a particular task on a particular processor, whereas the assignment of two communicating tasks to two processors involves a certain amount of costs, too, reflecting the energy involved in transferring data between the processors. The problem boils down to finding a "mapping" of the set of tasks to the set of processors that minimizes the total costs. We call this problem the MINIMUM WEIGHT PROCESSOR ASSIGNMENT problem (MWPA).

The paper is organized as follows.The next section gives a sketch of the background of the problem, and the motivation from current research in the area of Embedded Systems. The next section is followed by a section that provides the necessary definitions and gives a formal description of our model. In the fourth section we give a survey of NP-hard problems that all can be reduced to MWPA. We show that MWPA is already NP-hard if the set of processors is restricted to two, or if the (process) graphs are taken from the class of connected bipartite graphs with maximum degree at most 3. For the class of degree-2-contractible graphs we give a polynomial time algorithm. In the fifth section we generalize this algorithm into an exact (but not efficient) algorithm for any (process) graph.

2 Background and Motivation

2.1 General Background

Current research in the area of Embedded Systems is for a great deal driven by the explosive growth in the use of handheld mobile devices, such as cellular phones, personal digital assistants, digital cameras, global positioning systems, and so forth. Personal mobile computing (often referred to as ubiquitous computing) is likely to play a significant role in driving technology in the next decade. In this paradigm, the basic personal computing and communication device will be an integrated, battery-operated device, small enough to carry along all the time.

The technological developments needed to establish this paradigm of personal mobile computing lead to many challenging problems. In particular, these devices have limited battery resources, must handle diverse data types, and must operate in environments that are insecure, unplanned, and show different characteristics over time.

Traditionally, (embedded) systems that have demanding applications - e.g., driven by portability, performance, or cost - lead to the development of one or more custom processors or application-specific integrated circuits (ASICs) to meet the design objectives. However, the development of ASICs is expensive in time, manpower and money.

Another way to solve the problems has been to use general-purpose processors, i.e., trying to solve all kinds of applications running on a very high speed processor. A major drawback of using these general-purpose devices is that they are extremely inefficient in terms of utilising their resources.

In current developments the key issue to overcome the drawbacks of the aforementioned approaches is to use reconfigurable heterogeneous processors.

2.2 Current Research

To match the required computation with the architecture, in current research like the CHAMELEON project which is taking place at the University of Twente, an alternative approach is made in order to meet the requirements of future low-power hand-held systems. In the CHAMELEON project we apply reconfiguration at multiple levels of granularity. This means that the architecture contains different processing entities: e.g. a general-purpose processor (e.g. ARM core), a bit-level reconfigurable part (embedded FPGA) and several word-level reconfigurable parts (e.g. Montium tiles). The main philosophy used is that operations on data should be done at the place where it is most energy-efficient and where it minimizes the required communication. Partitioning is an important architectural decision, which dictates where applications can run, where data can be stored, and also the complexity of the mobile and the cost of communication services. Our approach is based on a dynamic (i.e. at run-time) matching of the architecture and the application.

2.3 Software Design Flow

The key issue in the design of portable multimedia systems is to find a good balance between flexibility and high-processing power on one side, and area and energy-efficiency of the implementation on the other side. The design of the above-mentioned architecture is useless without a proper tool chain supported by a solid design methodology. At various levels of abstraction, modern computing systems are defined in terms of processes and communication (or, at least, synchronisation) between processes. Many applications can be structured as a set of processes or threads that communicate via channels. These threads can be executed on various platforms (e.g. general purpose CPU, FPFA, FPGA, etc).

2.4 Process Graphs

We use a so-called Kahn based process graph model, which abstracts system functionality into a set of processes/tasks represented as vertices in a graph, and represents functional dependencies among processes (channels) with graph edges. The functionality of a process graph will be referred to as a set of (sub)tasks. This model emphasizes communication and concurrency between system processes. Edge labels are used to represent communication bandwidth requirements, while vertex labels may store a measure of process computational requirements. The costs associated with a process graph in the context of reconfiguration can be divided into communication costs between the processes, computational costs of the processes and initialization costs of the (sub)tasks. The costs can be expressed in terms of energy consumption, resource usage, and aspects of time

(latency, jitter, etc). The mapping of applications (a set of communicating tasks) is done in two phases. In the first phase (macro-mapping) for each set of tasks the optimal (or near to optimal) processing entity is determined. This phase defines what is processed where and when. In the second phase (micro-mapping) for each task a detailed mapping is derived to the platform of choice. The problem of this paper is related to the first phase.

2.5 Macro-mapping

In a reconfigurable system, application instantiation consists first of all of finding a suitable partition of the system specification into parts that can be mapped onto the most appropriate resources of the system (processors, FPGAs, coarse-grain reconfigurable entities). Because of the dynamics of the mobile environment we would like to perform the macro-mapping at run-time. The search for the 'best' mapping is typically a very hard problem, due to the size of the search space. We will refrain from giving more technical details, but stick to the (sub)problem of mapping the tasks to the heterogeneous processors in such a way that the costs of running the tasks on the processors and transferring data between communicating tasks is minimized.

3 Preliminaries

For the modeling of the optimization problem described in the previous section we consider simple graphs, denoted by $G = (V_G, E_G)$, where V_G is a finite nonempty set of vertices and E_G is a set of unordered pairs of vertices, called edges.

For a vertex $u \in V_G$ we denote its neighborhood, i.e. the set of adjacent vertices, by $N(u) = \{v \,|\, (u, v) \in E_G\}$. The *degree* $\deg(u)$ of a vertex u is the number of edges incident with it. The symbol Δ_G denotes the maximum degree among all vertices of G. If all vertices in G have the same degree $k \in \mathbb{N}$, then G is called *k-regular*.

A graph G is called *connected* if for every pair of distinct vertices u and v, there exists a *path* connecting u and v, i.e., a sequence of distinct vertices starting with u and ending with v where each pair of consecutive vertices forms an edge of G.

A graph is called *bipartite* if it is simple and its vertices can be partitioned into two sets A and B such that each edge has one of its end vertices in the set A and the other in B.

If $e = (u, v) \in E_G$, the *contraction* G/e of G is the simple graph obtained from G by replacing u and v and the edges incident with u and v by one new vertex uv and edges joining uv with the vertices adjacent to u or v in G. For another graph H, G is said to be *contractible* to H if H can be obtained from G by successive contractions of edges. A graph G is called *degree-2-contractible* if G is contractible to the graph consisting of one vertex by successively contracting

edges incident with vertices of degree 1 or 2. Note that the class of degree-2-contractible graphs is equal to the class of graphs that have tree-width at most two (cf. [2]).

Now let $G = (V_G, E_G)$ be a simple graph with vertex set V_G and edge set E_G. The vertices of G represent the tasks that have to be performed on a set P of processors. An edge $e = (u, v)$ exists if and only if there is communication between the processes of task u and task v.

Let the vertex weight $w_p^u \geq 0$ represent the costs of the process of task u, if u is performed on processor $p \in P$. This way we define a weight vector w^u of size $|P|$ for $u \in V_G$.

If in practice u cannot be performed on processor p, this can be expressed by setting $w_p^u = \infty$ (or a bounded, sufficiently large number M).

For $e = (u, v) \in E_G$ let the edge weight $w_{pq}^e \geq 0$ represent the communication costs between the processes of task u and v, if u is performed on processor $p \in P$ and v is performed on processor $q \in P$. This way we define a matrix W^e of size $|P| \times |P|$ for $e \in E_G$.

If in practice two tasks u and v cannot be performed on the same processor p simultaneously, we can model this by adding an edge $e = (u, v)$ and setting $w_{pp}^{(u,v)} = M$, where M is a sufficiently large number.

The graph G together with the weight vectors w^u, and weight matrices W^e is called a *weighted process graph*, and denoted by G_w.

We call a mapping $f : V_G \to P$ a *processor assignment* of G. Let $\mathcal{F}_G$ denote the set of all processor assignments of G. We define the *weight of a processor assignment* $f \in \mathcal{F}_G$ for a weighted process graph G_w as

$$w(f) = \sum_{v \in V_G} w_{f(v)}^v + \sum_{(u,v) \in E_G} w_{f(u)f(v)}^{(u,v)}.$$

A *minimum weight processor assignment* f^* is a processor assignment that has minimum weight, i.e., with $w(f^*) = \min\{w(f) \mid f \in \mathcal{F}_G\}$.

The Minimum Weight Processor Assignment problem (MWPA) is the problem of finding a minimum weight processor assignment for a given weighted process graph G_w and a set P of processors.

If any minimum weight processor assignment f will map task u on processor q, we say that u is *fixed* on q. If $w_p^u = 0$ for all processors $p \in P$, we will say that u is *free*.

4 Complexity Results

It is easy to prove that MWPA is an NP-hard problem. In fact several known NP-hard problems, such as the Minimum Multiway Cut problem ([3]), can be used to show this. We give the reduction from the Minimum Multiway Cut problem as an example. In later sections we show relations with other NP-hard problems which can be applied to show NP-hardness of MWPA for restricted graph classes, a limited number of processors, or arbitrarily close edge weights.

An instance of the MINIMUM MULTIWAY CUT problem is formed by an undirected graph G, a subset $S \subseteq V_G$ of terminals, and weights $c(e)$ for all $e \in E_G$. A *multiway cut* is a subset $F \subseteq E_G$ such that in the graph $G' = (V_G, E_G \backslash F)$ there is no path between any pair of terminals from S. The MINIMUM MULTIWAY CUT problem is the problem of finding a multiway cut F that minimizes the costs $c(F) = \sum_{e \in F} c(e)$ for a given weighted graph and a given set of terminals.

Observation 1 *The* MINIMUM MULTIWAY CUT *problem can be reduced to the* MINIMUM WEIGHT PROCESSOR ASSIGNMENT *problem.*

Proof. Given a graph G with a set $S = \{u_1, \ldots, u_s\}$ of terminals and weights $c(e)$ for $e \in E_G$ define a set of processors $P = \{p_1, \ldots, p_s\}$.

For $u \notin S$ let $w_p^u = 0$ for all $p \in P$. For $u_i \in S$ let $w_p^u = 0$ if $p = p_i$ and for $p \neq p_i$ let $w_p^u = M$, where M is a sufficiently large number. Then each minimum weight processor assignment fixes each terminal $u_i \in S$ to its 'own' processor p_i. For each $e \in E_G$ and $p, q \in P$ we define $w_{pq}^e = c(e)$ if $p \neq q$ and $w_{pq}^e = 0$ if $p = q$.

This way we have constructed an instance (G_w, P). Our claim is that finding a multiway cut with minimum costs comes down to solving MWPA on the instance (G_w, P).

Let f be a minimum weight processor assignment for (G_w, P). Then $F = \{e \in E_G \mid e = (u, v) \text{ with } f(u) \neq f(v)\}$ is a multiway cut with costs $c(F) = w(f)$. Suppose a multiway cut F' exists with $c(F') < c(F)$. We may assume that each vertex of $(V_G, E_G \backslash F')$ is connected to some u_i; otherwise there is a multiway cut $F'' \subset F'$ with $c(F'') \leq c(F')$. Now we define a processor assignment f' of (G_w, P) by $f'(u) = p_i$, if there exists a path from u to $u_i \in S$ in the graph $G' = (V_G, E_G \backslash F')$. Obviously, $w(f') = c(F') < w(f)$, a contradiction.

Hence solving MWPA is at least as hard as solving MINIMUM MULTIWAY CUT. □

In [3] it is shown that the MINIMUM MULTIWAY CUT problem is already NP-hard for the class of graphs with three terminals. For two terminals it is polynomially solvable by standard network flow techniques. In the sequel we prove NP-hardness of MWPA for instances with two processors, and for instances with only free vertices instead of $|V_G| - 3$ free vertices and 3 fixed vertices.

4.1 Processor Graphs with Only Free Vertices

We show that MWPA is still NP-hard if the instances (G_w, P) are restricted to the class of weighted process graphs with only free vertices. For this purpose we make a reduction from the well-known problem GRAPH k-COLORABILITY, which is known to be NP-complete for each integer $k \geq 3$ (cf [5]).

GRAPH k-COLORABILITY (GkC)
Instance: A graph G and a set of colors $C = \{a_1 \ldots a_k\}$.
Question: Is G k-colorable, i.e., does there exist a mapping $g : V_G \to C$ such that $g(u) \neq g(v)$ for all $u, v \in V_G$ with $(u, v) \in E_G$?

Proposition 1. *Let $\mathcal{G}$ be a class of graphs for which* GkC *is* NP*-complete. Then* MWPA *is already an* NP*-hard problem for instances (G_w, P) in which G is a graph from $\mathcal{G}$ with $|V_G|$ free vertices, and sets P of $|P| = k \geq 3$ processors.*

Proof. Given a graph G with color set $\{a_1, \ldots, a_k\}$ we define a set of processors $P = \{p_1, \ldots, p_k\}$. For each edge $e \in E_G$ we define weights $w^e_{p_i p_i} = 1$ for all $p_i \in P$ and $w^e_{p_i p_j} = 0$ if $i \neq j$. All weights w^u_p are set to 0. Obviously, G is k-colorable if and only if (G_w, P) has a minimum weight processor assignment f with $w(f) = 0$. □

GkC (with $k \geq 3$) is amongst others NP-complete for planar graphs and cographs (i.e., graphs which do not contain an induced path on 4 vertices). Hence, MWPA is NP-hard if instances (G_w, P) are restricted to one of these two classes (with $|P| \geq 3$).

Note that the above proof also shows that MWPA remains NP-hard when restricted to instances with only two different values for the edge weights (and no vertices with positive weights).

4.2 Sets of Exactly Two Processors

In Proposition 1 we have shown that MWPA is NP-hard for instances (G_w, P), where $|P| = k \geq 3$. Of course, the problem is trivially solvable in polynomial time if the set of processors only contains one processor. In this section we show that MWPA is NP-hard if the set of instances is restricted to instances with two processors. It turns out that the recently studied NP-hard problem Minimum Generalized Vertex Cover ([6]) is a special case in our model.

Instances of Minimum Generalized Vertex Cover are undirected graphs G with numbers $d_0(e) \geq d_1(e) \geq d_2(e) \geq 0$ for every edge $e \in E_G$, and costs $c(u)$ for every vertex $u \in V$. For $S \subseteq V_G$ and $e = (u, v) \in E_G$ define $c(e)$ to be the cost of e depending on the number of its end vertices that are included in S, i.e., $c(e) = d_0(e)$ if $u, v \in V_G \backslash S$, $c(e) = d_1(e)$ if $u \in S, v \in V_G \backslash S$ or $v \in S, u \in V_G \backslash S$, and $c(e) = d_2(e)$ if $u, v \in S$. A *generalized vertex cover* is a subset $S \subseteq V_G$. The Minimum Generalized Vertex Cover problem is the problem of finding a generalized vertex cover S that minimizes the costs $c(S) = \sum_{v \in S} c(u) + \sum_{e \in E_G} c(e)$ for a given graph G and cost function c.

By choosing $d_0(e) = 1$ and $d_1(e) = d_2(e) = 0$ for all $e \in E_G$, and $c(v) = \frac{1}{n}$ for all $v \in V_G$, it is clear that the Minimum Generalized Vertex Cover problem is a generalization of the well-known Minimum Vertex Cover problem. In the proof of the proposition below we show that our setting includes both problems.

Proposition 2. MWPA *is* NP*-hard, even if the class of instances (G_w, P) is restricted to instances with two processors.*

Proof. Assume we are given a graph G with costs $c(v)$ for $v \in V_G$, and numbers $d_i(e)$ for $e \in E_G$ and $0 \leq i \leq 2$. Define a set of processors $P = \{p, q\}$. Let $w^u_p := c(u)$ and $w^u_q := 0$ for all $u \in V_G$. Let $w^e_{pp} := d_2(e), w^e_{pq} = w^e_{qp} := d_1(e)$, and $w^e_{qq} := d_0(e)$ for all $e \in E_G$.

This way we have obtained an instance (G_w, P) of MWPA. Each processor assignment f corresponds to a specific generalized vertex cover S with costs $c(S) = w(f)$, namely $S = \{u \in V_G \mid f(u) = p\}$. Vice versa, each generalized vertex cover S corresponds to a unique processor assignment f with costs $w(f) = c(S)$: Choose f given by $f(u) = p$ if $u \in S$, and $f(u) = q$ if $u \notin S$.

Hence the Minimum Generalized Vertex Cover problem is a special case of the Minimum Weight Processor Assignment problem. □

4.3 Process Graphs with Arbitrarily Close Edge Weights

It is straightforward to see that MWPA is polynomially solvable for instances (G_w, P) with process graphs G_w with constant edge weights, or with edge weights $w^e_{pq} = w^u_p + w^v_q$ if $e = (u, v)$ and $p, q \in P$. In both cases a processor assignment f that maps each vertex u on a processor p for which w^u_p is minimal, is a minimum weight processor assignment.

However, from the proof of Proposition 1 it turns out that MWPA stays NP-hard if the class of instances is restricted to instances (G_w, P) with arbitrarily close weights: Choose in the proof of Proposition 1 weights $w^e_{p_i p_i} = \epsilon$, where $\epsilon > 0$ is an arbitrarily small value, and $w^e_{p_i p_j} = 0$ if $i \neq j$. As remarked before, this also implies NP-hardness of MWPA when restricted to instances with only two possible values for the edge weights.

4.4 Process Graphs That Are Degree-2-contractible

In the theorem below we show that MWPA can be solved in polynomial time for instances (G_w, P) where G is a degree-2-contractible graph. So MWPA can be solved efficiently if for instance G_w is a tree or a unicyclic graph. However, already for the class of 3-regular bipartite graphs the problem turns out to be NP-hard.

For the NP-hardness construction we make a reduction from a particular variant of Hypergraph 2-Colorability. This is a well-known NP-complete problem (cf. [5]).

Hypergraph 2-Colorability (H2C)
Instance: A set $Q = \{q_1, \ldots, q_m\}$ and a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ with $S_j \subseteq Q$ and $|S_j| = 3$ for $1 \leq j \leq n$.
Question: Is there a 2-coloring of $(Q, \mathcal{S})$, i.e., a partition of Q into $Q_1 \cup Q_2$ such that $Q_1 \cap S_j \neq \emptyset$ and $Q_2 \cap S_j \neq \emptyset$ for $1 \leq j \leq n$?

With such a hypergraph we associate its incidence graph I, which is a bipartite graph on $Q \cup \mathcal{S}$, where (q, S) forms an edge if and only if $q \in S$ (cf. Figure 1).

Note that we may assume without loss of generality that I is connected.

Theorem 1. MWPA *can be solved in polynomial time for instances* (G_w, P) *in which* G_w *is a degree-2-contractible graph.* MWPA *is already* NP*-hard if the class of instances only contains instances* (G_w, P)*, where* G_w *is a connected* 3*-regular bipartite graph.*

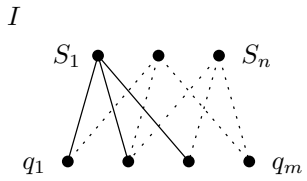

Fig. 1. The incidence graph of an instance of H2C

Proof. Let (G_w, P) form an instance of MWPA, where G is a degree-2-contractible graph. Suppose $v \in V_G$ and $\deg(v) = 1$. Let $u \in V_G$ be the only neighbor of v. Let $G' = G/(u,v) = (V_G \backslash v, E_G \backslash (u,v))$ be the graph G without the vertex v. For all $p \in P$ define

$$\tilde{w}_p^u = w_p^u + \min_{q \in P}\{w_q^v + w_{pq}^{(u,v)}\},$$

and do not change any other vertex or edge weights, i.e., $\tilde{w}_p^x = w_p^x$ for $x \in V_{G'} \backslash u, p \in P$ and $\tilde{w}_{pq}^e = w_{pq}^e$ for $e \in E_{G'}$ and $p, q \in P$. This way we have obtained an equivalent instance $(G'_{\tilde{w}}, P)$ of MWPA with one vertex less.

We continue with this procedure as long as we have a process graph with minimum degree equal to one. As long as we still have edges in our graph we proceed as follows.

Suppose $y \in V_G$ and $\deg(y) = 2$. Let $x, z \in V_G$ be the only two neighbors of v. We remove vertex y and edges (x,y) and (y,z), and add an edge (x,z) if it is not already in G. This way we have obtained a graph G^*. Assume $w_{pq}^{(x,z)} = 0$ for all $p, q \in P$ if (x,z) is not an edge in G. Then we define for all $p, q \in P$

$$\tilde{w}_{pq}^{(x,z)} = w_{pq}^{(x,z)} + \min_{r \in P}\{w_r^y + w_{pr}^{(x,y)} + w_{rq}^{(y,z)}\},$$

and do not change any other vertex or edge weights, i.e., $\tilde{w}_p^u = w_p^u$ for $u \in V_{G^*} \backslash y, p \in P$ and $\tilde{w}_{pq}^e = w_{pq}^e$ for $e \in E_{G^*} \backslash (x,z)$ and $p, q \in P$. This way we have obtained an equivalent instance $(G^*_{\tilde{w}}, P)$ of MWPA with one vertex less.

As long as G^* has edges, we repeat the steps above.

Combining these two steps gives us a polynomial time algorithm for degree-2-contractible graphs. Note that this approach fails, if during the procedure we obtain a graph $\bar{G}$ that does not have a vertex of degree one or two. (See the next section for a more general algorithm.)

We now prove NP-hardness for the class of instances (G_w, P) where G is a connected 3-regular bipartite graph.

Let $(Q, \mathcal{S})$ be an instance of H2C. First, we construct its incidence graph I. Since we assume that each set S_j has exactly 3 elements, we obtain that I is 3-regular. We introduce a set of processors

$$P = \{p_{qr} \mid (q,r) \in Q \times Q, q \neq r\} \bigcup \{p_q \mid q \in Q\} \bigcup \{p'_q \mid q \in Q\}.$$

Let M be a sufficiently large number. For each $S \in \mathcal{S}$ we define $w_p^S = 0$ if $p = p_{qr}$ and q, r are elements of S, and $w_p^S = M$ otherwise. For each $q \in Q$ we define $w_p^q = 0$ if $p = p_q$ or $p = p'_q$, and $w_p^q = M$ otherwise.

For each edge $e = (q, S) \in E_I$ we define the following edge weights. We make $w^e_{p_1 p_2} = 1$ if

- $p_1 = p_q$ and $p_2 = p_{rq}$ for some $r \in S$, or
- $p_1 = p'_q$ and $p_2 = p_{qr}$ for some $r \in S$.

We define $w^e_{p_1 p_2} = 0$ if

- $p_1 = p_q$ and $p_2 = p_{qr}$ for some $r \in S$, or
- $p_1 = p_q$ and $p_2 = p_{rt}$ for $r, t \in S \backslash q$, or
- $p_1 = p'_q$ and $p_2 = p_{rq}$ for some $r \in S$, or
- $p_1 = p'_q$ and $p_2 = p_{rt}$ for $r, t \in S \backslash q$.

Finally we define $w^e_{pq} = M$ for all other possibilities.

This way we have obtained a weighted process graph I_w. Our claim is that $(Q, \mathcal{S})$ is 2-colorable if and only if (I_w, P) has a minimum weight processor assignment f with $w(f) = 0$.

Suppose $Q_1 \cup Q_2$ is a 2-coloring of $(Q, \mathcal{S})$. If q is in Q_1 we define $f(q) = p_q$. Otherwise we let $f(q) = p'_q$. Since $Q_1 \cup Q_2$ is a 2-coloring, each $S \in \mathcal{S}$ contains a vertex q in Q_1 and a vertex r in Q_2. We define $f(S) = p_{qr}$. This way we have constructed a processor assignment f with $w(f) = 0$. Since all vertex and edge weights are positive, f is a minimum weight processor assignment.

To prove the reverse statement suppose a minimum weight processor assignment f exists with $w(f) = 0$. For all $q \in Q$ we do the following. If $f(q) = p_q$, we place q in Q_1. If $f(q) = p'_q$, then we place q in Q_2.

Now suppose $Q_1 \cup Q_2$ would not be a 2-coloring. Then a set $S = \{q, r, t\}$ in $\mathcal{S}$ exists that is fully contained in either Q_1 or Q_2. Suppose S is a subset of Q_1. By definition of Q_1, we obtain $f(q) = p_q$, $f(r) = p_r$, and $f(t) = p_t$ implying $w(f) > 0$. □

5 An Exact Algorithm

Although MWPA is NP-hard for many graph classes, here we present a relatively simple algorithm that determines the weight of a minimum processor assignment for *any* weighted process graph G_w and set of processors P. We call this algorithm MINWEIGHT. Its running time is exponential. However, in practice it could compute solutions quite fast, as long as the input graphs have a small number of vertices with a high degree (greater than two) or a high number of fixed vertices.

Steps (3) and (4) in the algorithm have already been explained in the proof of Theorem 1. Step (5) handles vertices with degree greater than two. Each time this step is executed the number of computations increases exponentially.

Note that, in practice rules such as "delete forbidden processor/vertex combinations" can be added to increase the running time of MINWEIGHT in case there are many of such combinations. It is also easy to implement the algorithm in such a way that it gives as output a minimum weight processor assignment f with $w(f) = w^*$. Moreover, instead of using the same (Cartesian products of)

processor sets for all vertices, in an implementation of the algorithm we could use different sets for different vertices in order to save on memory and computational steps.

Algorithm MINWEIGHT

(**1**) FOR $(u,v) \notin E_G$ and $p,q \in P$ DO $w_{pq}^{(u,v)} := 0$.

(**2**) Choose a vertex $v \in V_G$.

(**3**) IF $\deg(v) = 1$,
THEN let $V_G := V_G \backslash v$, and $E_G := E_G \backslash (u,v)$ for $u \in N(v)$.
FOR $u \in N(v)$ and $p \in P$ DO

$$w_p^u := w_p^u + \min_{q \in P}\{w_q^v + w_{pq}^{(u,v)}\}.$$

(**4**) IF $\deg(v) = 2$,
THEN let $V_G := V_G \backslash v$ and $E_G := (E_G \cup \{(x,z)\}) \backslash \{(x,v),(v,z)\}$ for $x,z \in N(v)$.
FOR $x,z \in N(v)$ and $p,q \in P$ DO

$$w_{pq}^{(x,z)} := w_{pq}^{(x,z)} + \min_{r \in P}\{w_r^v + w_{pr}^{(x,v)} + w_{rq}^{(v,z)}\}.$$

(**5**) IF $\deg(v) \geq 3$,
THEN choose a vertex $u \in N(v)$. Let $G := G/(u,v)$.
Set $P := P \times P$.
FOR $(p,q) \in P$ DO $w_{(p,q)}^{uv} := w_p^u + w_{pq}^{(u,v)} + w_q^v$.
FOR $x \in V_G$ and $(p,q) \in P$ DO $w_{(p,q)}^x := w_p^x$.
FOR $e = (uv,x)$ with $x \in N(uv)$ and $(p,q),(r,s) \in P$ DO $w_{(p,q)(r,s)}^{(uv,x)} := w_{pr}^{(u,x)} + w_{qr}^{(v,x)}$.
FOR $e \in E_G$ not incident with uv and $(p,q),(r,s) \in P$ DO $w_{(p,q)(r,s)}^e := w_{pr}^e$.

(**6**) IF $|V_G| \geq 2$, THEN GOTO (2).

(**7**) Output $w^* := \min\{w_p^u \mid p \in P, u \in V_G\}$. STOP.

---

6 Conclusion

In the previous sections we have introduced the Minimum Weight Processor Assignment problem (MWPA), motivated from current research in the area of Embedded Systems. We have analysed the complexity of MWPA, its relation to various other graph problems, and showed that it is NP-hard, even when restricted to very specific classes of instances. For a number of classes of instances we have shown that MWPA is polynomial. We presented a simple exact (exponential) algorithm, based on edge contractions, for solving MWPA for general instances. We are currently investigating the running-time of the algorithm for real-life examples. After an extensive search initiated by a remark of Hans Bodlaender we became aware of a number of related papers. In some of them, the static assignment problem has been studied under different names, whereas this paper is motivated by the dynamic problem of allocating tasks in run-time. Part

of the results of the present paper are covered. We refer the interested reader to [4] and the survey paper [1].

Acknowledgements. The authors thank Asaf Levin for explaining the relationship of the problem to the minimum multiway cut problem.

References

1. Aardal, K. I., van Hoesel, S. P. M., Koster, A. M. C. A., Mannino, C., Sassano, A. Models and solution techniques for frequency assignment problems. *4OR 1,* 4 (2003), 261–317.
2. Bodlaender A tourist guide through treewidth. *Acta Cybernetica 11* (1993), 1–21.
3. Dahlhaus, E., Johnson, D. S., Papadimitriou, C. H., Seymour, P. D., and Yannakakis, M. The complexity of multiterminal cuts. *SIAM Journal on Computing 23,* 4 (1994), 864–894.
4. Fernández-Baca, D. Allocating modules to processors in a distributed system. *IEEE Transactions on Software Engineering 15,* 11 (1989), 1427–1436.
5. Garey, M. R., and Johnson, D. S. *Computers and Intractability.* W. H. Freeman and Co., New York, 1979.
6. Hassin, R., and Levin, A. The minimum generalized vertex cover problem. In *Algorithms ESA 2003, 11th ESA '03, Budapest, Hungary* (2003), no. 2832 in Lecture Notes in Computer Science, Springer Verlag, 289 – 300.
7. Smit, G.J.M., Havinga, P.J.M., Smit, L.T., Heysters, P.M., and Rosien, M.A.J. Dynamic reconfiguration in mobile systems. In *Field-Programmable Logic and Applications. Reconfigurable Computing Is Going Mainstream, 12th FPL '02, Montpellier, France* (2002), no. 2438 in Lecture Notes in Computer Science, Springer Verlag, 171–181.