5 Automatische partitionering van softwaresystemen

Rob Bisseling, Jarosław Byrka, Selin Cerav-Erbas, Nebojša Gvozdenović, Mathias Lorenz, Rudi Pendavingh, Colin Reeves, Matthias Röger en Arie Verhoeven

Kostenbesparende gepartitioneerde softwaresystemen

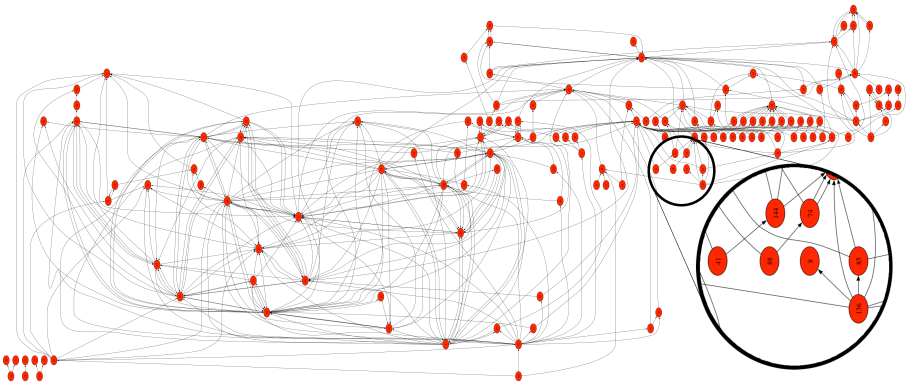
De afgelopen jaren heeft de vooruitgang in de informatietechnologie een enorme invloed gehad. Steeds meer organisaties, zoals banken en overheidsinstellingen, zijn afhankelijk van immer groeiende softwaresystemen om hun taken te kunnen uitvoeren. De grote complexiteit van deze systemen maakt het systeemonderhoud erg duur. Een mogelijke oplossing voor dit probleem bestaat in het opsplitsen van deze systemen in kleinere minder complexe modules. Iedere module wordt dan onderhouden door een kleine groep systeembeheerders. Zo'n gepartitioneerd systeem heeft echter een *interface* nodig voor de communicatie tussen de modules. Dit betekent dat ieder programma dat door andere modules gebruikt wordt, ook wel een interfaceprogramma genoemd, voorzien wordt van extra communicatiefaciliteiten. De kosten voor het extra onderhoud van deze interface zijn evenredig met de grootte van de interface ofwel het totaal aantal interfaceprogramma's. Voor een goede partitionering is de omvang van de interface minimaal en zijn de modules voldoende klein. In de praktijk zijn voor het vinden van zo'n splitsing ervaren deskundigen nodig. Voor erg grote softwaresystemen is het echter bijzonder handig om gebruik te maken van automatische partitioneeralgoritmen, die een goede suggestie kunnen geven. Het bedrijf Software Improvement Group (SIG, `http://www.sig.nl`) gevestigd in Diemen ontwikkelt dit soort hulpmiddelen.

Partitieprobleem van een call-graaf

Een *graaf* is een bekend wiskundig concept voor het representeren van een netwerk. Een graaf bestaat uit een aantal punten, *knopen* genaamd, die verbonden zijn door lijnen. Als de verbindingen ook een richting hebben, worden deze *pijlen* genoemd en spreken we van een gerichte graaf. Een softwaresysteem kan worden gemodelleerd door een call-graaf, waarvan de knopen de programma's voorstellen. Als programma v programma w aanroept (een call) is dit equivalent met een pijl van knoop v naar knoop w. Een korte notatie hiervoor is $v \to w$. Een module kan nu worden gerepresenteerd door een deelverzameling van de knopen van de call-graaf. Dit betekent dat de omvang van een module gelijk is aan het aantal knopen van de bijbehorende deelverzameling. We noemen knoop v een *interfaceknoop* als hij een inkomende pijl heeft vanuit een knoop in een andere module. De grootte van de interface is nu gelijk aan het aantal interfaceknopen. Zodoende kan het vinden van een goede splitsing voor het softwaresysteem worden geformuleerd als het volgende partitioneringsprobleem van een call-graaf:

Gegeven: een call-graaf en de gehele getallen K, L.

Vind: een partitionering van de knopen van de call-graaf in L deelverzamelingen zodat iedere deelverzameling maximaal K knopen heeft en het totaal aantal interfaceknopen minimaal is.



Figuur 5.1: Call-graaf van het softwaresysteem `Java1` met 158 knopen (programma's). De inzet rechtsonder geeft een gedetailleerde blik op een deel van de graaf waar de pijlen tussen de programma's duidelijk zichtbaar zijn.

Figuur 5.1 laat een call-graaf zien van een softwaresysteem uit de praktijk met de naam `Java1`.

Verschillende oplosmethoden

Twee verschillende probleemformuleringen blijken veelbelovend voor het probleem van SIG. Allereerst modelleren we het probleem als een *geheeltallig lineair programmeringsprobleem* (in het Engels: *Integer Linear Programming*, ILP). We gebruiken daarbij beslissingsvariabelen y_{vl} en x_{vl}, waarvan een waardering als volgt moet worden opgevat:

$$y_{vl} = \begin{cases} 1 & \text{knoop } v \text{ zit in deelverzameling } l \\ 0 & \text{anders} \end{cases}$$

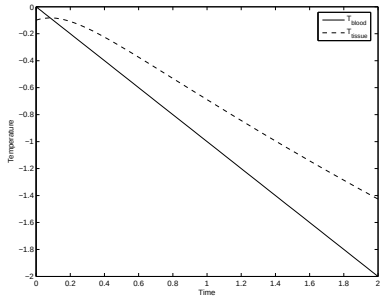
$$x_{vl} = \begin{cases} 1 & \text{knoop } v \text{ is een interfaceknoop} \\ & \text{bevat in deelverzameling } l \\ 0 & \text{anders.} \end{cases}$$

De eis dat module l niet meer dan K knopen mag bevatten kan dan geformuleerd worden als een lineaire ongelijkheid:

$$y_{v_1 l} + y_{v_2 l} + \cdots + y_{v_N l} \leq K.$$

En evenzo kan met lineaire vergelijkingen worden afgedwongen dat elke knoop in precies één module zit, etc. Het doel om zo min mogelijk interfaceknopen te gebruiken komt neer op het minimaliseren van de som van alle x_{vl}.

Deze formulering is een goede eerste stap naar een praktische oplossing. Een ILP probleem heeft een sterk meetkundig karakter, en er is een effective oplossingsmethode die de meetkundige aard van dit probleem uitbuit. Zodoende kan ook ons partitioneringsprobleem exact worden opgelost, maar de rekentijd loopt snel op als de grootte van het softwaresysteem toeneemt. Efficiënte herformuleringen en slimme hulpalgoritmen worden cruciaal om de oplossing te vinden binnen een redelijke tijd.



Figuur 5.2: Verbindingsmatrix (158×158) van het probleem `Java1`.

Als alternatief hebben we het probleem ook geformuleerd als een *hypergraafpartitioneringsprobleem*. Een hypergraaf is een generalisatie van een gewone ongerichte graaf met netten in plaats van lijnen. Een *net* verbindt een willekeurig aantal knopen, terwijl een lijn slechts twee knopen verbindt. Net n_j bestaat uit knoop j en alle knopen i met $i \rightarrow j$. Als de knopen in net n_j in verschillende deelverzamelingen vallen, is het net *gebroken*. Na toevoeging van de pijl $j \rightarrow j$ voor alle knopen j, blijkt het aantal interfaceknopen gelijk te zijn aan het aantal gebroken netten. Het call-graafpartitioneringsprobleem kan dus ook worden geformuleerd als een hypergraafpartitioneringsprobleem (HP).

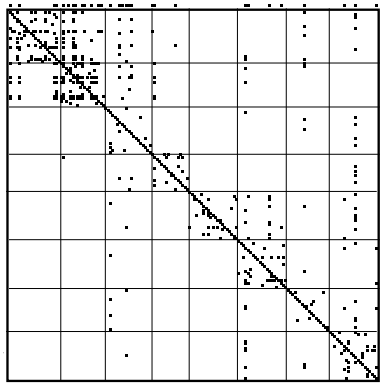
Een gerichte graaf met N knopen wordt gewoonlijk beschreven door haar verbindingsmatrix $A = (a_{ij})_{i,j=1,\ldots,N}$, gedefiniëerd door

$$a_{ij} = \begin{cases} 1 & \text{als } i \rightarrow j, \\ 0 & \text{anders.} \end{cases}$$

Figuur 5.2 laat de verbindingsmatrix zien die hoort bij de call-graaf `Java1`. De verbindingsmatrix kan worden gepartitioneerd door zogenoemde multi-level methoden, zoals het matrix-partitioneringsprogramma Mondriaan. Dit programma werd aangepast, zodat het toegepast kon worden op de voorbeelden van SIG. Het is *heuristisch* (niet exact) van aard, maar desondanks bleek het erg snel en betrouwbaar zodat de resultaten dicht bij de optimale oplossing liggen.

Vergelijking van de twee algoritmen

We hebben twee methoden ontwikkeld voor het probleem, namelijk een optimaal ILP-en een heuristisch HP-algoritme. Het is een interessante vraag hoe de kwaliteit en rekentijd zich verhouden tussen deze twee varianten. Daarom zijn voor een aantal praktijkproblemen, geleverd door SIG, de beide methoden gebruikt om een optimale partitionering in $L = 8$ modules te vinden, waarbij de maximale omvang van een module gelijk is aan $K = 1.2\frac{N}{L}$. Het softwarepakket CPLEX is gebruikt op een 400 MHz Sun Ultra-Sparc II computer om de optimale oplossing te vinden, terwijl Mondriaan een 867 MHz Apple PowerBook G4 computer gebruikte. Figuur 5.3 laat het resultaat van het hypergraafpartitioneringsalgoritme zien voor het probleem `Java1`. Elke kolom correspondeert met een net; gebroken netten zijn bovenaan gemarkeerd.



Figuur 5.3: Gepartitioneerde verbindingsmatrix (158×158) van het probleem `Java1`. De rijen en kolommen zijn zodanig gepermuteerd dat ieder rij-blok bij één module hoort.

In Tabellen 5.1 en 5.2 is de interfaceomvang $|I|$ en de rekentijd weergegeven voor beide methoden. De verschillende call-grafen hebben N knopen en $|A|$ pijlen. Als we de resultaten vergelijken zien we dat de heuristische methode hele goede resultaten levert. De ILP methode leidt tot de optimale interface; de interface verkregen door de heuristische HP methode is groter, echter hoogstens een factor 1.63. Aan de andere kant is de rekentijd van de HP methode veel kleiner en lineair evenredig met $|A|$. De rekentijd van de ILP methode is over het algemeen veel hoger en heeft een onvoorspelbaar karakter.

Conclusie

Het splitsen van een groot softwaresysteem in kleinere eenheden is voor veel organisaties een uitdaging geworden. Tijdens de "Study Group Mathematics with Industry"maakten we kennis met dit probleem, waar het werd geïntroduceerd door SIG. Tijdens de week van de studiegroep en erna hebben we gewerkt aan een duidelijke wiskundige formulering van het probleem. Tevens hebben we twee verschillende aan-

Probleem	N	$\|A\|$	K	L	opt. $\|I\|$	$\|I\|/N$ (%)	rekentijd (s)
`Java1`	144	422	23	8	26	16.5	103
`Java3`	837	5252	127	8	242	28.4	246456^a
`Java4`	15	39	2	8	11	68.8	0.22
`Cobol1`	947	1900	209	8	13	0.9	118
`Cobol2`	449	659	81	8	6	1.1	351
`Cobol3`	1145	2686	203	8	51	3.8	6452
`Cobol4`	1100	2951	167	8	32	2.9	742172^a

Tabel 5.1: Resultaten met 0-1 lineair programmeren. De met 'a' gemarkeerde tijden zijn afkomstig van een andere computer.

Probleem	N	$\|A\|$	beste $\|I\|$	gem. $\|I\|$	$\|I\|/N$ (%)	rekentijd (s)
`Java1`	158	580	30	30.7	19.0	0.06
`Java3`	851	6103	275	283.2	32.3	0.54
`Java4`	16	55	11	11.2	68.8	0.001
`Cobol1`	1398	3298	17	22.4	1.2	0.33
`Cobol2`	545	1204	10	11.5	1.8	0.12
`Cobol3`	1357	4043	69	74.6	5.1	0.34
`Cobol4`	1116	4067	52	56.5	4.7	0.41

Tabel 5.2: Resultaten met hypergraafpartitionering.

pakken gevonden, die beide erg geschikt en waardevol kunnen zijn voor de bedoelde toepassingen. De wiskundige formuleringen reduceerden het probleem tot standaardproblemen uit de discrete optimalisering. Zodoende konden moderne algoritmen gebruikt worden om de praktijkproblemen van SIG op te lossen. Onze twee methoden kunnen het relatieve aantal interfaceprogramma's drastisch verlagen. Voor Cobolsystemen is het zelfs mogelijk dat de omvang van de interface minder dan 5% wordt. Voor kleine problemen bevelen wij aan de optimale ILP methode te gebruiken. Voor grote problemen, met duizenden knopen in de call-graaf, is multilevel hypergraafpartitionering de enige realistische optie.

Het bedrijf SIG waardeerde de voorgestelde oplossingen, evenals het inzicht dat verkregen kon worden door middel van ons onderzoek. Aan de andere kant hebben wij geprofiteerd van dit goed gestelde probleem wat ons leidde tot nieuwe interessante vragen zoals de vergelijking tussen een exacte en een heuristische methode in een realistische situatie.