Om (prestanda)utvärdering av algoritmer och datastrukturer

Institutionen för Elektronik, programvaru och datorsystem
Robert Rönngren 070227

1.0 Om prestandautvärderingar

Prestandautvärderingar av algoritmer och program är en viktig del i många verksamheter. Det kan handla om att utvärdera algoritmer för att hitta bästa algoritmen vid design/utveckling av en produkt, att jämföra olika programvara vid upphandlingar eller helt enkelt att redogöra för nya algoritmers prestanda jämfört med befintliga i ett forskningssammanhang. Oavsett vad syftet eller sammanhanget är så har dessa utvärderingar det gemensamt att de bör planeras, genomföras och utvärderas på ett vetenskapligt/ingenjörsmässigt sätt. De grunder man har att utgå från vid utvärdering av algoritmer och programvara är de samma som används för alla typer av vetenskapliga utvärderingar/analyser/experiment. Dessa grundläggande kunskaper i experimentell metodik bör vara en del av varje ingenjörs baskunskaper. Skälet till detta är naturligtvis att det i ingenjörens arbetsuppgifter ofta ingår att genomföra rättvisande och objektiva utvärderingar att ha som underlag för olika typer av beslut.

För de flesta typer av utvärderingar gäller att det ofta är ett både komplext och tidsödande arbete att genomföra en bra utvärdering. Som alltid när man ställs inför en sådan uppgift kunde man önska att det fanns en enkel väg ut ur problemen, gärna i form av en “kokboks”-metod. Nu är det tyvärr sällan så och även i det här fallet finns inget enkelt recept att följa som passar i alla fall. Däremot finns det några saker man bör tänka på, några steg som man i det flesta fall måste gå igenom och en övergripande metod att angripa problemet med.

1.1 Olika typer av utvärderingar

Olika utvärderingar skiljer sig naturligtvis från varandra i detaljer, men har det gemensamt att de grovt kan klassificeras som: *analytiska*, *empiriska* (genom experiment) eller baserade på en kombination av dessa tillvägagångssätt. Ofta är det fördelaktigt att kombinera analytiska och experimentella metoder. Till exempel kan experimentella resultat valideras mot analytiska egenskaper. Om de experimentella resultaten inte överensstämmer med de analytiska måste man naturligtvis söka rimliga skäl till detta samt utreda om de experimentella resultaten är behäftade med någon typ av fel.

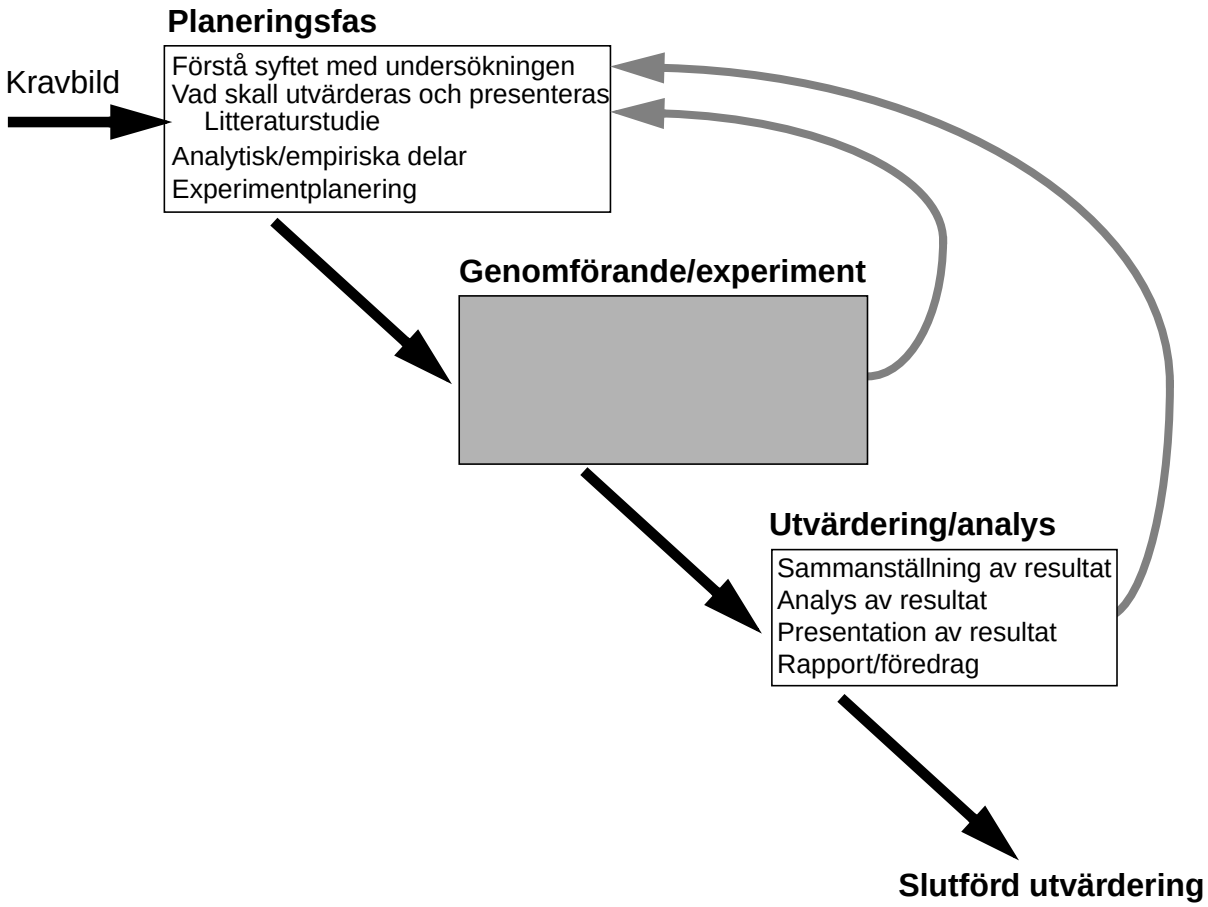
Själva genomförandet av en utvärdering kan normalt delas in i tre faser:

- **Planering**
- **Genomförande (experiment)**
- **Utvärdering/analys**

Om de här faserna kan sägas att de sällan är av samma omfattning. I praktiken lägger man oftast mest tid och omsorg på, i fallande ordning: Planering, Utvärdering/analys, Genomförande/experiment. Om inte annat kommer du att märka detta genom omfånget på beskrivningen av de olika delarna.

Ofta tvingas man också iterera över de olika faserna, ibland på grund av bristande planläggning som kan göra att man måste gå tillbaka och utföra fler experiment, men ofta som ett resultat av att man i analysen av resultaten upptäckt saker som behöver undersökas noggrannare.

I den modell vi föreslår för utvärdering av algoritmer och datastrukturer på datorsystem, se Figur 1, gör vi en vidare uppdelning av de olika faserna. Även om det specifikt rör en viss typ av prestandautvärderingar kan man applicera den föreslagna strukturen på ett brett område av undersökningar.



FIGUR 1. Schematisk bild över de föreslagna stegen i en (prestanda)utvärdering

2.0 Planeringsfasen

Planeringsfasen kan delas upp i två huvuddelar:

- Förstå/bestämma/avgränsa syftet med utvärderingen
- Planering av genomförandefasen och analysfasen

2.1 Om vikten av att själv förstå syftet med utvärderingen

Att ha en klar och tydlig förståelse för vad syftet med utvärderingen är och hur utvärderingen kommer att användas är helt avgörande för att kunna göra en bra utvärdering. Syftet med utvärderingen har direkt påverkan på hur utvärderingen bäst genomförs, t.ex. vilka experiment som behöver genomföras för att man skall kunna besvara de frågor som utvärderingen skall kunna besvara.

Förenklat kan man dela upp utvärderingar i två typer vad avser syftet:

- Generell utvärdering
- Utvärdering för ett specifikt syfte

2.1.1 Generella utvärderingar

I en generell utvärdering strävar man efter att belysa så många relevanta aspekter av ett system eller en algoritm som möjligt. Typiskt är att man planerar både sina teoretiska analyser och experiment så att man kan belysa ett brett spektrum av möjliga relevanta arbetslaster och förhållanden som systemet/algoritmen kan komma att användas i. För- och nackdelar bör också tydligt belysas och lyftas fram. Det kan t.ex handla om under vilka förhållanden som systemet/algoritmen får optimal prestanda och under vilka förhållanden prestanda blir som sämst.

Man bör också sträva efter att göra jämförelser med andra relevanta system och algoritmer för att underlätta för den som tar del av utvärderingen att själv kunna förstå systemets/algoritmens för- och nackdelar. Detta gäller speciellt experimentella utvärderingar där det kan vara svårt, för att inte säga direkt omöjligt, att jämföra t.ex exekveringstider för två olika algoritmer om de är uppmätta på olika system. Att direkt jämföra exekveringstider från experiment med en algoritm på t.ex en PC med en Intel 386i processor med körningar gjorda med en annan algoritm på t.ex en PC utrustad med en Pentium processor går inte. Har man däremot utvärderat bägge algoritmerna på samma målsystem (i vårt fall datorystem) är det möjligt att direkt kunna jämföra dess egenskaper och dra mer generella slutsatser av experimenten.

En generell utvärdering leder ofta till att experimenten och analysen av systemet blir mer omfattande än en mer specifik utvärdering.

För att samanfatta så bör en generell utvärdering belysa det utvärderade systemets egenskaper både analytiskt och experimentellt vad avser:

- För- och nackdelar (“best/worst case analysis/experiments”)
- Troliga arbetslaster/användningsfall
- Omfatta en jämförelse med relevanta liknande algoritmer/system

2.1.2 Utvärderingar med ett specifikt syfte

För en utvärdering med ett specifikt syfte gäller att man måste skaffa sig en tydlig bild av den miljö och speciella förhållanden som utvärderingen skall gälla i. För utvärdering av algoritmer kan det t.ex handla om på vilken typ av datorsystem algoritmen skall exekveras och vilken typ av indata man kommer att ha. Men principiellt sett så skiljer sig inte en utvärdering med ett specifikt syfte från en mer generell mer än att man i oftast kan begränsa utvärderingen, experimenten och analysen till att gälla inom de begränsningar som ges. Man bör naturligtvis fortfarande belysa både bästa, sämsta och troliga fall för systemen/algoritmerna både analytiskt och experimentellt.

2.2 Att planera utvärderingen

När man väl förstått syftet med utvärderingen kan man planera hur utvärderingen skall genomföras. Detta är normalt en process som sker i flera steg.

Steg 1: *Identifiera vad som skall utvärderas och hur resultaten skall presenteras*

För att kunna planera en utvärdering på ett bra sätt måste man på ett tidigt stadium identifiera vad man vill att utvärderingen skall resultera i. Det vill säga man måste tidigt skaffa sig ett bra grepp om vilken typ av resultat man vill kunna presentera för att undersökningen skall fylla sitt syfte. Utgående från det planerar man sedan genomförandet och analysen av undersökningen.

Till underlag för det här arbetet kan man dels ha krav som ställs direkt eller indirekt från en eventuell uppdragsgivare som t.ex. kanske kräver att en viss algoritm skall utvärderas med avseende på vissa egenskaper. Man har också mycket att vinna på att göra en noggrann ***litteraturstudie***. I många sammanhang kan man hitta sk. "benchmarks" som används för jämförande utvärderingar. Dessa kan innefatta både algoritmer att jämföra med, indata och vilken typ av analys/experiment som skall utföras. Att använda vanligt förekommande benchmarks som en del av utvärderingen gör att man lättare kan jämföra och validera resultaten mot andra, liknande, undersökningar om sådana finns. I de flesta sammanhang är det också av avgörande betydelse för kvaliteten på en utvärdering att den som utför utvärderingen själv förstår hur och i vilka sammanhang det man skall utvärdera kan komma att användas. Inte minst kan det underlätta experimentplaneringen.

Rent konkret bör man tidigt identifiera vilka system/algoritmer som är de huvudsakliga kandidaterna för utvärderingen och vilka som eventuellt behöver finnas med i utvärderingen för jämförelse. Man måste också bestämma vilka egenskaper hos systemen/algoritmerna/datastrukturerna som skall utvärderas.

Egenskaper som kan vara relevanta att studera när det gäller utvärdering av algoritmer/datastrukturer kan beroende på vilken typ av algoritm/datastruktur det handlar om t.ex. vara:

- minnesförbrukning och exekveringstid som en funktion av indata
- känslighet för indata (t.ex bästa/värsta/troliga fall av indata)
- tid för konvergens (dvs tid till dess man kommer till ett stabilt tillstånd, hittar en lösning etc.)
- kvalitet på lösning (t.ex för algoritmer som söker en nära, men kanske inte garanterat, optimal lösning på ett problem)

Den här fasen skall identifiera vilken typ av resultat som man behöver för analysen, vilket i sin tur styr vilken typ av experiment som behöver utföras.

Steg 2: *Identifiera vad som kan/bör utvärderas analytiskt respektive experimentellt*

Här bör en grundregel vara att om man kan bygga (enkla) analytiska modeller som täcker in de viktigaste aspekterna av det system/den algoritm man vill studera så bör man också göra det. De analytiska modellerna bör man generellt sett ta fram innan man utför experimenten. För en eventuell experimentell del av utvärderingen har man nytta av de analytiska modellerna dels för att de kan användas för att validera de mätresultat experimenten genererar, men också för att de kan utgöra en grund för planeringen av vilka experiment man skall genomföra.

> ***Om analytiska modeller:*** *Analytiska modeller används i de flesta naturvetenskapliga sammanhang. Ett exemplet, och något som man borde kunna förvänta sig att alla ingenjörsstuderande känner igen från gymnasiet, är Newtons mekanik. Newtons mekanik är en förenkling av verkligheten som vi känner till den idag, dvs en förenkling gentemot Einsteins teorier. Det som gör Newtons mekanik till ett bra exempel på en analytisk modell är att det är en användbar förenk-*

ling av verkligeten som ger oss tillräcklig precision, dvs återspeglar verkligheten tillräckligt väl, för att modellen skall vara praktiskt användbar inom sina tillämpningsområden. Modellen kan också jämföras med verkligheten, dvs. ***valideras*** *(värderas) mot verkligheten, genom att man experimentellt kan se att modellen stämmer tillräckligt bra med verkligheten. Att en anlytisk modell inte behöver vara exakt in i minsta detalj,* ***men att den måste vara tillräckligt exakt för att kunna användas den tänkta tillämpningen****, är en viktig observation.*

Här är det på sin plats att introducera begreppen ***verifiering*** *och* ***validering*** *och klargöra skillnaden mellan dem.*

- *Verifiering innebär att man kan* ***bevisa*** *en egenskap, t.ex korrektheten, hos ett system/en algoritm.*
- *Validering innebär att man försöker* ***värdera korrektheten/giltigheten*** *hos till exempel mätningar på ett system.*

Ofta innebär validering att man jämför mot något känt och försöker uppskatta rimligheten hos det man validerar med detta som måttstock. Validering är användbart i många sammanhang då man inte kan verifiera korrektheten hos ett system.

Ett för kursen närliggande exempel på en analytisk modell är att modellera tidskomplexiteten, dvs den exekveringstid man kan förvänta sig går åt, för operationer på prioritetsköer. En prioritetskö är en kö med (åtminstonde) två operationer:

1) Att ta ut elementet med högst prioritet;

2) Att sätta in ett element med (godtycklig) prioritet i kön.

En enkel implementation av en prioritetskö skulle kunna baseras på en sorterad dubbellänkad lista där elementen sorteras in vid insättning så att alla element ligger i prioritetsordning i listan. Tidskomplexiteten uttryckt i ***stora ordo notation*** *det vill säga att man anger den dominerande termen för komplexiteten utan hänsyn till konstanter etc., är för uttagsoperationen O(1) (dvs det kan göras i konstant tid) och för insättningen O(n) där n är antalet element i listan. Dvs insättningens tidskomplexitet växer linjärt med antalet element i listan. Studerar man en användning av en dubbellänkad lista där man kombinerar både insättningar och uttag ur listan så bestäms tidskomplexiteten för detta av den operation som har den snabbast växande tidskomplexiteten, dvs den blir för en godtycklig operation O(n).*

En relevant fråga är om man kan ha någon nytta av så grova modeller som ett stora ordo mått på tidskomplexitet? Naturligtvis är det användbart i många sammanhang. Till exempel kan man validera experimentella resultat. Om man mäter tidskomplexiteten för insättningar/uttag i en prioritetskö implementerad som en dubbellänkad lista och den uppmätta tiden inte växer som O(n) (dvs linjärt med antalet element i listan) utan t.ex snabbare, så måste man fundera på om det är fel i den analytiska modellen eller, troligare, i mätningarna eller i implementationen av listan.

Man kan också ha nytta av så enkla modeller som ordo notation för komplexitet för att kunna jämföra olika typer av algoritmer. En annan klass av datastrukturer som lämpar sig väl för implementation av prioritetsköer är trädstrukturer. I ett balanserat träd är tidskomplexiteten för insättning av ett element normalt O(log(n)) och samma gäller för uttag av ett element. För en godtycklig operation blir då tidskomplexiteten också O(log(n)). Jämför man den dubbellänkade listans tidskomplexitet med en trädstrukturs kan man förvänta sig att trädstrukturen är överlägsen den länkade listan då antalet element i prioritetskön är stort. Däremot är en begränsning hos stora ordo notationen att man inte kan säga något om vilken datastruktiur som är snabbast för små prioritetsköer. Det bestäms i huvudsak av tidskonstanter som ordo notationen inte tar hänsyn till. I själva verket visar det sig att länkade listor oftast är snabbare än trädstrukturer i praktiken om antalet element i dem är litet (<~100 element).

Den här typen av analyser är också användbara när man försöker analysera värsta respektive bästa fall för algoritmer. För den dubbellänkade listan kan man enkelt konstruera användningsfall där fördelningen av prioriteter på de element man sätter in är sådan att varje nytt element man sätter in har lägre prioritet än vad som finns i listan. I insättningsoperationen kan man då välja att sätta in nya element bakifrån. I det här specifika fallet slipper man då sortera in elementen vid insättning vilket gör att insättningen får en tidskomplexitet som är O(1). Denna typ av analyser resulterar ofta i att man designar experiment för att explicit visa på bästa/värsta fall beteenden. Exempelvis kan man konstruera fall av användning av vissa typer av trädstrukturer som gör att dessa degenererar till att bara en gren växer, dvs att de degenererar till vad som i praktiken motsvarar en länkad lista. I dessa fall blir tidskomplexiteten för en godtycklig operation inte längre O(log(n)) utan O(n), vilket man bör kunna visa med väldesignade experiment.

Amorterad komplexitet: *Amorterad komplexitet är ett begrepp som är användbart i många sammanhang där t.ex operationer på en datastruktur eller i en algoritm kan ta väldigt olika lång tid att utföra. Låt oss exemplifiera begreppet genom att studera datastrukturer för att implementera prioritetsköer. Ofta utförs sorteringen av elementen i en prioritetskö vid insättning och i vissa fall vid uttag av varje element, jfr. en länkad lista där hela sorteringen sköts vid varje insättning eller en trädstruktur där sortering normalt sker vid både insättning och uttag. En observation som gjorts är att man borde kunna designa datastrukturer som skjuter upp sorterandet av elementen i datastrukturen till en tidpunkt då de verkligen behöver sorteras (exempel på sådana datastrukturer är t.ex Calendar queue och Lazy queue). Tanken är att skjuta upp sorteringen av elementen till dess att de behöver vara sorterade och på så sätt spara tid. Det vill säga att man låter sorteringen ske till följd av en uttagsoperation och bara i de fall då det är nödvändigt, dvs. då man inte har direkt tillgång till elementet med högst prioritet. Implementerar man en prioritetskö på detta sätt kan man göra insättningarna (osorterat) på konstant tid, dvs med tidskomplexitet O(1) och förhoppningsvis de flesta uttagsoperationerna i konstant tid (i alla de fall där elementet med högst prioritet finns direkt tillgängligt). Däremot kommer vissa uttagsoperationer att ta betydligt längre tid när man tvingas göra en sortering. I de fall då man gör en sortering kan komplexiteten för uttagsoperationen vara O(nlog(n)) eller kanske högre som $O(n^2)$ beroende på tidskomplexiteten hos den algoritm som används för sorteringen. Men om man kan visa att man bara utför sortering i t.ex högst var n:te uttagsoperation kan man amortera kostnaden för sorteringen över n opertioner och på så sätt få en amorterad tidskomplexitet som är t.ex O(log(n)) (eller O(n)). Som kuriosa kan nämnas att det existerar prioritetsköer baserade på sådana ideer som för de flesta vanliga prioritetsfördelningar har O(1) prestanda för både insättningar och uttag.*

Ett annat exempel där man utnyttjar att man kan amortera komplexa operationer över ett antal enklare är i sidindelat virtuellminne där man amorterar kostbara minnesreferenser, dvs. de som resulterar i sidfel, över en stor mängd minnsesreferenser som inte generera sidfel. För den typen av system är det dock svårt att bevisa mer generellt att man får låg tidskomplexitet. Till exempel kan de flesta system tvingas in i "thrashing"-beteenden om man överlastar dem.

Steg 3: *Planera eventuella experiment*

När man vet vad man vill åstadkomma med utvärderingen och man identifierat att vissa delar behöver utvärderas experimentellt så är det dags att planera experimenten. Det man bör ha som ledstjärna när man designar sina experiment är att de i alla fall måste uppfylla de mest elementära kraven på "vetenskaplighet" för att resultaten skall ha något värde. Grundkraven som måste uppfyllas är:

- ***att man mätt rätt saker (både vad gäller att man mätt det som är relevant och att man i mätningen verkligen mätt vad man tror sig mäta)***
- ***att man vet och anger vilken noggrannhet man har på mätningarna***
- ***att man valt experiment och dokumenterat dem så att de kan återupprepas (reproduceras)***

De här kraven kan tyckas vara självklarheter men ofta kan det vara svårt att få en tillräckligt väl kontrollerad experimentuppställning för att kunna säkerställa dem och i allt för många fall kan man se att den som gjort utvärderingen ignorerat dessa krav (då blir man inte godkänd på laborationerna i den här kursen....)

I vetenskapliga sammanhang väljer man som regel att göra rådata allmänt tillgängligt men också källkod till de algoritmer/system man utvärderat, allt för att uppfylla kraven på reproducerbarhet.

I planeringen av experimenten är det i huvudsak två saker att tänka på:

- Experimentuppställning (plattform)
- Experimenten i sig
-

Experimentuppställning

I experimentuppställningen ingår allt man behöver för att kunna utföra experimenten och de mätningar man vill göra. Om man gör en prestandautvärdering av programvara så ingår i experimentuppställningen: datorer med minnessystem och operativsystem etc., run-time system, implementationen av de algoritmer man utvärderar, hur de är kompilerade, ev. andra processer som kör på datorn man utför mätningarna på mm. I detta ryms många olika parametrar och faktorer som kan påverka utvärderingen. Därför är det ofta också svårt att direkt jämföra två utvärderingar utförda på olika experimentuppställningar.

I valet av experimentuppställning bör man välja en uppställning som så väl som möjligt avspeglar syftet med utvärderingen samtidigt som man försöka minimera osäkerhets- och felfaktorer. Exempelvis är det kanske inte så värdefullt att utvärdera en algoritm som avses köras på ett litet inbyggt datorsystem på den mest högpresterande server man har tillgäng till.

Finns inga andra krav bör datorsystemen man använder vara så moderna som möjligt och så lite lastade som möjligt. Kompilering bör vara utförd så att man utnyttjar maximal optimering. För generella utvärderingar kan det vara intressant att i vissa fall utnyttja olika datorsystem för att få en bild av hur t.ex. cache- och minneskonfiguration, processorer, operativsystem och kompilatorer påverkar prestandan.

I experimentuppställningen ingår också de metoder man använder för att göra mätningarna av t.ex exekveringstid och minnesförbrukning.

Experimentuppställningen måste naturligtvis noggrannt dokumenteras och redovisas i utvärderingen för att utvärderingen skall bli reproducerbar, men också för att den som tar del av utvärderingen själv skall kunna bilda sig en bild av vad som påverkat den uppmätta prestandan.

Experiment

För experimentplaneringen gällar att man bör identifiera vilka system/algoritmer man vill mäta på, vilka parametrar man vill variera, hur dessa skall varieras och vilka mätdata som skall samlas in. Man måste också ägna mycket tanke åt hur man skapar sig en uppfattning om tillförlitligheten i resultaten, t.ex hur många gånger ett enskilt experiment måste upprepas för att man via mätdata skall kunna bilda sig en korrekt feluppskattning. All planering syftar naturligtvis till att få ut *alla* de resultat man behöver till utvärderings/analys fasen så snabbt och enkelt som möjligt, gärna i ett svep från genomförandefasen. Speciellt viktig är planeringen i de fall man inte har ständig och enkel tillgång till experimentplattformen, vilket kan vara fallet om den t.ex måste bokas i förväg. Lämpligt är också i de flesta fall att bara variera en av parametrarna i taget i experimenten för att resultaten skall bli lättare att tolka.

För att konkretisera detta kan vi fortsätta med vårt exempel som handlar om utvärdering av algoritmer/datastrukturer för implementation av prioritetsköer. För en sådan utvärdering kan man enkelt identifiera ett antal parametrar som kan varieras som:

- *Algoritmer/datastrukturer*
- *Användningsfall/arbetslast inkluderat fördelning på prioriterna hos elementen som sätts in i prioritetskön*
- *Antal element i prioritetskön*

Prestanda kan i det här fallet lämpligen mätas som exekveringstid och minnesförbrukning. När det gäller användningsfall/arbetslaster för en prioritetskö kan man strukturera upp dem som bestående av två delar:

1) Antalet element i prioritetskön och hur detta antal varierar. Variationen av antalet element i kön kan renodlas till tre fall - i) man gör bara insättningar och storleken på kön växer, ii) man gör bara uttag ur kön och dess storlek krymper, iii) kön befinner sig i ett "steady-state" och antalet uttag/insättningar av element balanserar varandra så attantalet element (storleken) är konstant.

2) Den andra delen av arbetslasten utgörs av fördelningen av prioriteterna hos elementen som sätts in i kön.

Hur man väljer parametrarna till arbetslasten och hur de skall varieras beror naturligtvis på syftet med undersökningen. I en specialiserad undersökning kan man ibland begränsa variationen av parametrarna mer än i en generell undersökning där man normalt bör undersöka så stor del av parameterrymden som möjligt (dvs så många rimliga kombinationer av algoritmer/datastrukturer, köstorlekar, prioritetsfördelningar som möjligt och också tydligt peka på styrkor/begränsningar hos de olika algoritmerna)

3.0 Genomförande (experiment)

Genomförandedelen bör vara den enklaste delen av utvärderingen. Går allt som det ska så utför man de experiment man planerat och samlar in de mätdata som planerats. Det man bör försöka kontrollera är eventuella felkällor, t.ex att inte andra processer startas på plattformen. Vad gäller själva mätningarna så kan det vara på sin plats att disskutera hur man bäst gör mätningar av tids- repektive minnesförbrukning.

3.1 Tidsmätningar

Om vi skall göra tidsmätningar i ett experiment som omfattar ett antal operationer på en datastruktur/algoritm har vi i princip två val:

- Mäta tiden för varje operation individuellt
- Mäta tiden över alla operationer

Att mäta på individuella operationer

För att kunna mäta tiden på individuella operationer krävs att man har tillgång till en klocka med tillräcklig upplösning så att mätningarna blir tillräckligt exakta för sitt syfte. Till exempel är det inte är lönt att försöka mäta förlopp som tar några enstaka sekunder med klockor som har en upplösning på minuter eller sekunder. För att i det fallet få tillförlitliga och rättvisande mätningar måste man kunna mäta tid i tiondels eller hundradels sekunder.

En fördel med att mäta på individuella operationer är att man undviker att mäta tid för exekveringen av resten av programmet som ofta är en slinga där man även gör andra beräkningar än de operationer man egentligen är intresserad att mäta på.

Att mäta tid över alla operationer

Mäter man tiden över alla operationer kommer man inte bara att mäta tiden för operationerna i sig men också tiden som går åt för exekveringen av det testprogram som operationerna utförs i. Naturligtvis bör man försöka renodla testprogrammen så att man utför så lite onödigt arbete som möjligt utöver de operationer man är intresserad av att mäta. Ett problem som ändå kan uppstå är att tiden som åtgår för att exekvera övriga delar i testprogrammet döljer prestanda hos det man är intresserad att mäta. Ett konkret exempel: Om man mäter tiden över ett helt testprogram för ett antal operationer på en datastruktur A och får resultatet 5 tidsenheter och tiden för att utföra samma operationer på en annan datastruktur B är 6 tidsenheter, så går det inte direkt att säga hur långsamma operationerna på datastruktur B är i förhållande till operationerna på datastruktur A (dvs. är datastruktur B 10, 20 eller kanske 30% långsammare än A?). Den typen av analys går bara att göra om man vet hur stor del av den uppmätta exekveringstiden som utgjordes av de operationer man är intresserad av. För att uppskatta tiden som spenderas i de operationer man är intresserad av kan man göra på följande sätt: man mäter tiden över testprogrammet utan de operationer man är intresserad av och subtraherar den tiden från tider som uppmäts i experiment där man utför tidsmätning av testprogrammet med de operationer man vill mäta på. Om man har kontroll på eventuella felkällor kan denna metod vara en väg att mer exakt kunna uppskatta tidsförbrukningen för de operationer man är intresserad av. Å andra sidan kan man få systematiska fel om endera tidsmätningen är behäftad med fel.

Felkällor i tidsmätningar på datorsystem

I alla tidsmätningar finns felkällor. Speciellt gäller för tidsmätningar på datorsystem att det ofta finns felkällor som kan vara svåra att kontrollera. Oftast uppstår dessa fel som en följd av att andra processer direkt eller indirekt stör tidsmätningarna. Om andra användarprocesser eller processer i OS:et stör exekveringen av testprogrammet kan man råka ut för att man mäter tid då testprogrammet inte exekverar för att det ligger i ready-kön eller för att det i onödan väntar på I/O-enheter som andra processer använder. Indirekt kan andra processer också orsaka störningar. I fallet att man har en global sidutbytesalgoritm kan t.ex. en annan process förorsaka extra sidfel för den process man mäter på vilket i sin tur påverkar exekveringstiden.

Den här typen av fel kan vara svåra att undvika om man inte kan ha full kontroll över experimentplattformen själv. Men om man inte kan kontrollera felen kan man i alla fall försöka minska risken att få problem genom att t.ex försöka kontrollera att ingen annan kör på experimentplattformen samtidigt som man kör sina experiment genom att kontrollera vilka som är inloggade på platformen och/eller genom att se vilka andra processer som är aktiva (t.ex genom att kontrollera med kommandot `top` före och efter exekvering av testerna). Det här gäller kanske speciellt för dig som student eftersom du oftast inte kan hindra andra att logga in på gemensamma

datorresurser eller stänga av systemtjänster som har daemoner/processer igång.

Att skilja ut felkällor från indatakänslighet

När man mäter tidsåtgång för exekvering av algoritmer/datastrukturer så får man ofta en viss variation i tidsåtgången. Den här variationen kan bestå av två komponenter som kan vara svåra att särskilja om man inte ägnar viss eftertanke åt att designa experimenten. Komponenterna är:

i) fel från olika typer av felkällor;

ii) algoritmens/datastrukturens känslighet för indata.

För att kunna skaffa sig en rättvisande bild av hur stor inverkan dessa två källor till variation har på resultaten måste man ofta utföra flera typer av experiment. Låt oss återigen illustrera detta med exmplet att mäta tidsåtgång för insättningar/uttag av element i prioritetsköer. Antag att vi mäter exekveringstiden för att göra N insättnings- och uttagsoperationer på en prioritetskö i steady-state (dsv då antalet element i kön är konstant). Prioriteterna på de element som sätts in i kön genereras av en pseudoslumptalsfunktion f(X) där X är det initial slumptalsfröet. För att kunna bilda sig en uppfattning om både mätfel och indatakänslighet måste man i detta fall utföra (minst) två uppsättningar experiment:

i) En uppskattning av mätfelen kan fås genom att upprepa samma experiment med samma initiala slumptalsfrö ett antal gånger för att se hur de exekveringstider man mäter varierar. All variation i mätningarna beror i detta fall på osäkerhet (fel) i mätningarna.

ii) För att kunna bilda sig en uppfattning om datastrukturens känslighet för indata så genomför man istället ett antal mätningar med olika startvärden för det initiala slumptalsfröet. Känner man då till mätfelen kan man uppskatta algoritmens/datastrukturens känsligheten för indata som funktion av slumptalsfröet (De flesta väldesignade algoritmer/datastrukturer för prioritetsköer borde vara tämligen okänsliga för just detta fall, men alla är det inte. Vanligare är större känslighet om man byter fördelning på prioriteterna).

Hade man istället direkt utfört ett antal mätningar där man varierat slumptalsfröet (utan att först bildat sig en uppfattning om mätfelen) hade man inte kunnat särskilja mätfelen från känsligheten för indata.

Om indata och slumptalsgeneratorer

Indata till experiment är normalt antingen indata från “verkliga” system eller syntetiskt genererad indata. Om man t.ex vill utvärdera olika sidutbytesalgoritmer i ett operativsystem kan det vara lämpligt att använda indata för vilka sidor som refereras i exekvering av riktiga program. Men i många fall används indata som genereras med hjälp av slumptalsgeneratorer.

Ordet slumptalsgenerator är nästan undantagslöst missvisande, istället bör man använda ordet pseudoslumptalsgenerator. En sådan generator är en matematisk algoritm som utgående från ett intialt värde (frö, eng. seed) genererar en ström av tal som bör ha egenskapen att om man analyserar strömmen med avseende på dess statistiska egenskaper så ska den uppvisa ett beteende som är svårt att skilja från en analys av en ström av tal från den stokastiska fördelning man försöker efterlikna. Kvaliten på den ström av tal som genereras av en pseudoslumptalsgenerator beror dels på kvaliten på algoritmen i sig men också på hur intialvärdet (fröet) väljs. Ibland förekommer också andra typer av problem som att i vissa fall kan strömmen av tal som genereras visa upp goda statistiska egenskaper om man analyserar den tal för tal (i en dimension) men om man väljer att analysera par av tal från strömmen, för att t.ex spänna upp två dimensioner, så kan man i vissa fall få dåliga statistiska egenskaper.

Om detta ämne finns mycket som kan sägas, men det ligger utanför vad som avhandlas i den här kursen. Den intresserade rekommenderas istället att läsa en kurs i t.ex Modellering och simule-

ring eller läsa vidare i litteratur som “Simulation modeling & analysis” av Law och Kelton McGraw&Hills förlag.

Mätning av minnesåtgång

Mätning av minnesförbrulkning går, till skillnad mot tidsmätningar, att utföra exakt. Dock måste man vara noggrann med hur man skall gå till väga. Mäter man på ett minneshanteringsbibliotek och dess minnesförbrukning (som är det sammanhang den här skriften är tänkt att användas i) kan man på ett POSIX-kompatibelt system läsa av minnesförbrukningen genom att med hjälp av `brk()/sbrk()` se hur brytpunkten för heapen förflyttats. Att använda det sättet att mäta minnesförbrukningen för ett “vanligt” program leder till felaktiga mätdata eftersom man då i huvudsak kommer att mäta minnesförbrukningen hos algoritmerna i minneshanteringsbiblioteket som ofta förallokerar stora datablock av prestandaskäl. För den typen av mätningar måste man istället instrumentera (förändra) programmet så att man räknar hur mycket minne (antal bytes) som de operationer man vill mäta på allokerar/avallokerar. Exempel på hur man gör i bägge fallen hittar du i testprogrammen till minnesallokeringslaborationen i OS-kursen.

4.0 Utvärdering/analys

Efter genomförande/experimentfasen tar utvärderings/analysfasen vid. Även denna fas kan delas upp i tre delar:

- Sammanställning av resultat
- Analys av resultat
- Presentation av resultat

4.1 Sammanställning av resultat

Sammanställning av resultaten är ofta ett rättframt arbete. Vanligen innebär det att man beräknar medelvärden, gör feluppskattningar och sammanställer resultaten i form av grafer (kurvor) eller i diagram för att underlätta den fortsatta analysen. Just feluppskattningar är en ofta försummad detalj som vi har anledning att återkomma till.

4.2 Analys av resultat

När man sammanställt resultaten på en form som underlättar tolkningen av dem vidtar analysfasen. Har man analytiska modeller för det man mätt experimentellt så jämför man de experimentella resultaten med de analytiska, dvs man validerar de experimentella resultaten mot de analytiska. Om dessa inte stämmer överens måste man gå djupare i analysen för att söka finna en trovärdig förklaring till skillnaderna. I huvudsak kan orsakerna bakom att experimentdata inte överenstämmer med vad de analytiska modellerna förutsäger vara:

- Något saknas eller är fel i den analytiska modellen som då bör förfinas/korrigeras
- Man har felkällor i mätningarna som man inte kunnat kontrollera, i vilket fall man bör göra om mätningarna under mer kontrollerade former
- Man har mätt effekter som inte enbart kan härledas till algoritmen/datastrukturerna man utvärderar men som kan bero på hur dessa samverkar med det underliggande datorsystemet

Att kunna identifiera effekter som beror på det senare fallet är ett problem som kräver god kännedom om hela experimentuppställningen. Tidsmätningar av exekvering av algoritmer/operationer på datastrukturer på ett datorsystem innebär nästan undantagslöst att man mäter de kombinerade effekterna av den algoritm/datastruktur man mäter på och hur dessa samverkar med det underliggande datorsystemet. Exempel på effekter man kan råka ut för är t.ex att cachebeteendet fungerar bra för små datamängder men sämre för stora vilket kan resultera i “knän” på kurvor, eller att liknande effekter uppkommer om virtuellminnessystemen börjar fungera dåligt (dvs. då många sid-

fel genereras) för mycket stora datamängder. Noterar man sådana effekter tvingas man ofta planera (nya) experiment för att verifiera/validera hypoteserna.

Ibland upptäcker man också i analysen enstaka mätpunkter som kraftigt avviker från det förväntade. I dessa fall bör man, om man har möjlighet, exakt återupprepa den mätning eller det experiment som gav upphov till det avvikande resultatet. Om resultaten av de återupprepade körningarna bättre överensstämmer med de förväntade resultaten finns fog att anta att de avvikande resultaten orsakades av mätfel. Endast i detta fall, dvs att man kan säkerställa att ett mätfel ligger bakom resultaten, kan “felaktiga/konstiga” mätresultat avskrivas. I annat fall bör man redovisa mätresultaten, om möjligt tillsammans med en djupare analys av vad det är som orsakat det/de avvikande mätresultaten.

Slutresultatet av analysen bör alltid vara att man kan ge *väl underbyggda förklaringar till alla resultat* och att man, sett till syftet med undersökningen, inte saknar väsentliga mätresultat/analyser.

4.3 Presentation av resultaten

För presentationen av resultaten/utvärderingen bör följande gälla oavsett om det sker i en form av en rapport eller i form av en muntlig presentation:

- **Presentationen av resultaten skall hjälpa läsaren/åhöraren att förstå och rätt kunna tolka resultaten/utvärderingen**
- **Experiment och experimentplattform skall redovisas i rapporten och(/eller) göras tillgängliga så att experimenten kan reproduceras**
- **Alla resultat och slutsatser måste ges rimliga förklaringar/tolkningar, skulle det vara så olyckligt att man inte kan ge ett resultat en rimlig förklaring måste det redovisas öppet. Man får aldrig försöka dölja “konstiga” resultat.**
- **Figurer, grafer och tabeller skall vara läsliga och “self-contained”**
  Exempelvis har en graf tämligen begränsat värde om man inte tydligt kan utläsa de kurvor som finns i den (utan att behöva lupp eller att ha den i fyrfärg). Att en figur, graf eller tabell är “self-contained” innebär att den skall innehålla förklarande text så att man tillsammans med tabell- eller figurtext kan förstå vad axlar, rader och kolumner representerar, vilka enheter som används, vad olika kurvor representerar samt vilka experiment med tillhörande parametrar det är som redovisas i tabellen, figuren eller grafen utan att man behöver läsa annan text eller, än värre, att man tvingas försöka gissa sig till vad det är som redovisas.
- **Alla experimentella resultat måste redovisas med noggrannt angivna felmarginaler**

Den sista punkten är något som det allt för ofta syndas mot. Delvis kan det bero på att många system för att t.ex rita grafer inte enkelt tillåter att man redovisar felmarginaler i graferna. För att förstå varför man inte kan slarva med att ange felmarginaler så kan vi betrakta följande exempel: Du skall välja en algoritm för en implementation av en tjänst där kraven är att man skall ha låg och förutsägbar exekveringstid. Medelexekveringstiden för algoritm A är 10 tidsenheter och för algoritm B 12 tidsenheter. Kan du mot bakgrund av de redovisade värdena dra slutsatsen att algoritm A är att föredra? Svaret på den frågan är entydigt NEJ! Utan att känna till felmarginalerna/noggrannheten i mätvärdena kan man inte dra några slutsatser alls. Anta till exempel att man redovisar ett absolutfel för algoritm A som är +/- 5 tidsenheter och att absolutfelet för algoritm B är +/- 1 tidsenhet. Med den kännedomen är sannolikt algoritm B att föredra framför algoritm A.

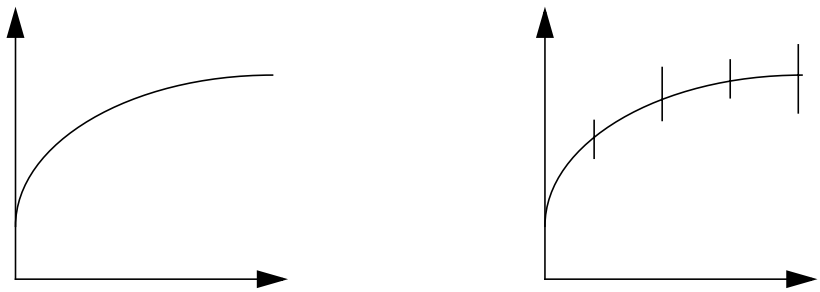
Att redovisa felmarginaler/noggrannhet

I huvudsak kan felmarginaler/noggrannhet redovisas som:

- Konfidensintervall
- Absolutfel
- Relativfel - i de fall felet är linjärt beroende av den storhet man mätt
- Varians

Vilket sätt man väljer beror på omständigheterna. Själv föredrar författaren att använda konfidensintervall och/eller absolutfel. Den som glömt hur man beräknar konfidensintervall, varians etc. uppmanas att leta fram litteraturen från någon kurs i matematisk statistik som läsaren förväntas ha studerat!

Att redovisa felmarginaler/noggrannhet i tabellform är tämligen rättframt, i grafer kan det däremot vara svårare. Har man bara en kurva i en graf kan man oftast enkelt redovisa felmarginaler/noggranhet i form av “felstolpar” kring mätpunkterna i kurvan/grafen, se Figur 2. Redovisar man resultat från flera experiment, t.ex i form av flera kurvor, i en och samma graf kan det däremot vara svårt att i grafen redovisa felmarginalerna då den snabbt kan bli oläslig. Då får man redovisa felmarginaler/noggrannhet separat i t.ex en tabell eller om omständigheterna så medger i form av relativfel.



FIGUR 2. Exempel på graf utan respektive med “felstolpar”

5.0 Slutord

Om du läst så här långt och förväntat dig att hitta hur du exakt skall gå till väga för att göra en specifik prestandautvärdering på en specifik experimentuppställning, som t.ex att minnesförbrukning och tidskomplexitet för biblioteksrutinerna `malloc()`, `free()` och `realloc()`, så lär du bli besviken. Exakt hur man bäst går till väga för att göra utvärderingar på olika typer av målsystem och för olika algoritmer/datastrukturer går inte att beskriva i en mer allmänt hållen text. Men tar du de allmänna råd som getts i den här texten på allvar så riskerar du i alla fall inte att grunden för din prestantautvärdering blir dålig oavsett vad du utvärderar.