Obliczenia równoległe w General Game Playing

Maciej Świechowski
promotor: prof. dr hab. Jacek Mańdziuk

Plan

- Krótkie przypomnienie **GGP/MiNI-Player**
- Motywacja
- Pomysł 2012
- Pomysł 2013
- Porównanie
- Podsumowanie

General Game Playing [GGP]

Idea, aby jeden gracz grał w wiele gier

- Uniwersalne metody

  (nauki, reprezentacji wiedzy, abstrakcyjnego rozumowania, wybierania akcji)

- Powrót do podstaw AI – przeniesienie odpowiedzialności za stosowanie inteligentnych algorytmów analizy gier z programisty na program

General Game Playing [GGP]

General Game Playing Competition

- Nazwa konkursu organizowanego przez *Stanford Logic Group* od 2005 roku, w ramach konferencji **AAAI** (IJCAI w 2011)
- Obiektywne środowisko testowe
- Nagroda 10 000 USD dla zwycięzcy

Mistrzowie świata GGP

2005: ***Cluneplayer***, USA

2006: ***Fluxplayer***, Niemcy

2007: ***CadiaPlayer***, Islandia

2008: ***CadiaPlayer***, Islandia

2009: ***Ary***, Francja

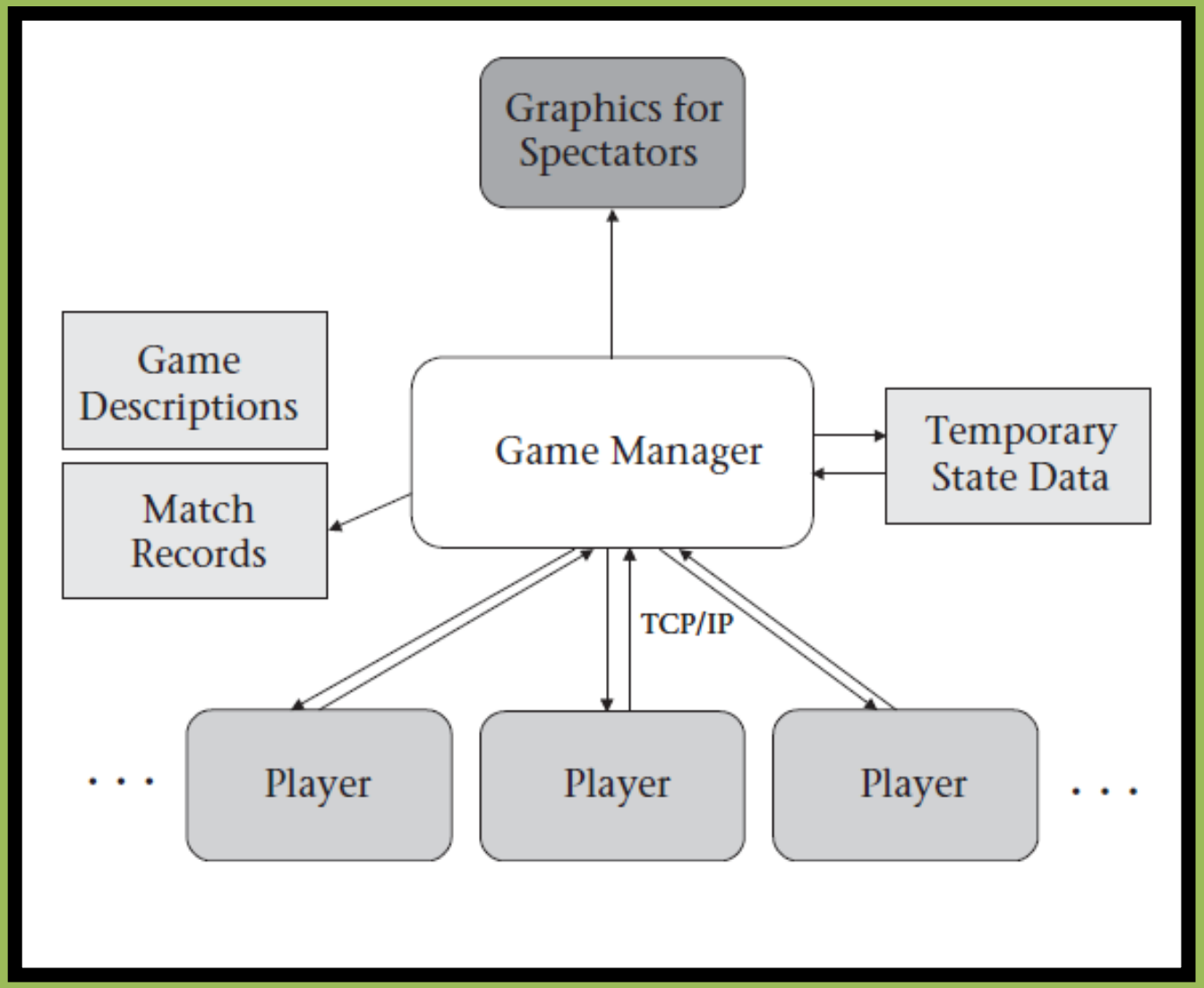
2010: ***Ary***, Francja

2011: ***TurboTurtle***, USA

2012: ***CadiaPlayer***, Islandia

era Monte-Carlo

General Game Playing



Komunikacja:

- **START**
- **PLAY**
- STOP
- ABORT, PING

Limity czasowe:

- **START CLOCK**
- **PLAY CLOCK**

Wymiana komunikatów

Game Manager Message	Game Player Response
(START MATCH.435 WHITE *description* 90 30)	READY
(PLAY MATCH.435 (NIL NIL))	(MARK 2 2)
(PLAY MATCH.435 ((MARK 2 2) NOOP))	NOOP
(PLAY MATCH.435 (NOOP (MARK 1 3))	(MARK 1 2)
(PLAY MATCH.435 ((MARK 1 2) NOOP))	NOOP
. . .	. . .
(STOP MATCH.435 ((MARK 3 3) NOOP)	DONE

Język opisu reguł gier

Game Description Language (GDL)

- Język oparty na logice pierwszego rzędu
- Wariant ***Dataloga***, wielu uczestników korzysta z kompilatorów ogólniejszego języka: ***Prologa***

Klasa reprezentowanych gier (GDL-I):

- **skończone**
- **deterministyczne**
- synchroniczne

MiNI-Player

MiNI-Player

Agent GGP

- wystawiany przez nas

Brał udział w konkursie **GGP Competition 2012**

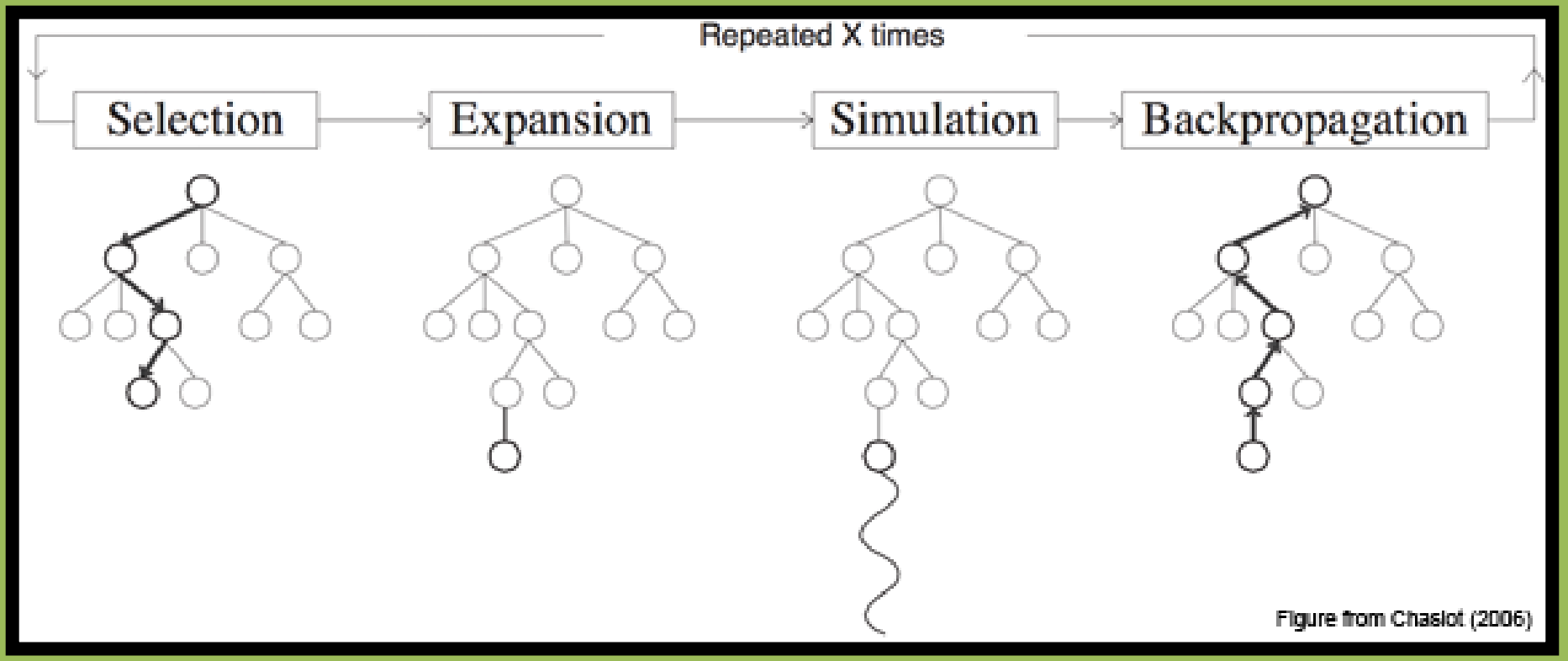
Szczegółowy opis: seminarium 12/03

„MiNI-Player – podsumowanie dotychczasowych badań”

MiNI-Player – główne aspekty

- Interpretacja reguł
- Algorytmy przeszukiwania drzewa: MCTS
  - Wybór kontynuacji podczas konstrukcji drzewa
  - Wybór akcji do zagrania
- Strategie gry (Monte-Carlo playout policies)
- Moduł wyboru strategii
- Zrównoleglenie
- Komunikacja

MiNI-Player – główne aspekty



MiNI-Player – główne aspekty

Selekcja: w każdym węźle rozpoczynając od wierzchołka wybieramy najlepszego potomka do kontynuacji przeszukiwania (tu: algorytm UCT).

Rozwinięcie: tworzymy nowy węzeł

Symulacja: gramy zgodnie z wyznaczoną do symulacji strategią aż do osiągnięcia stanu terminalnego

Propagacja wsteczna: uaktualniamy statystyki – średni wynik oraz liczbę odwiedzin w każdym węźle na ścieżce, którą wybraliśmy

Motywacja

zrównoleglania obliczeń

Po co zrównoleglać? (5 powodów)

1. Większość programów wykorzystuje przeszukiwanie (konstrukcję) drzewa gry.

Ilość odwiedzonych stanów jest ważnym wyznacznikiem jakości gry – zwłaszcza w podejściach typu Monte-Carlo.

Podejścia symulacyjne: im więcej przeprowadzonych symulacji, tym bardziej adekwatna ocena akcji/stanów.

Po co zrównoleglać?

2. Wszyscy zrównoleglają...

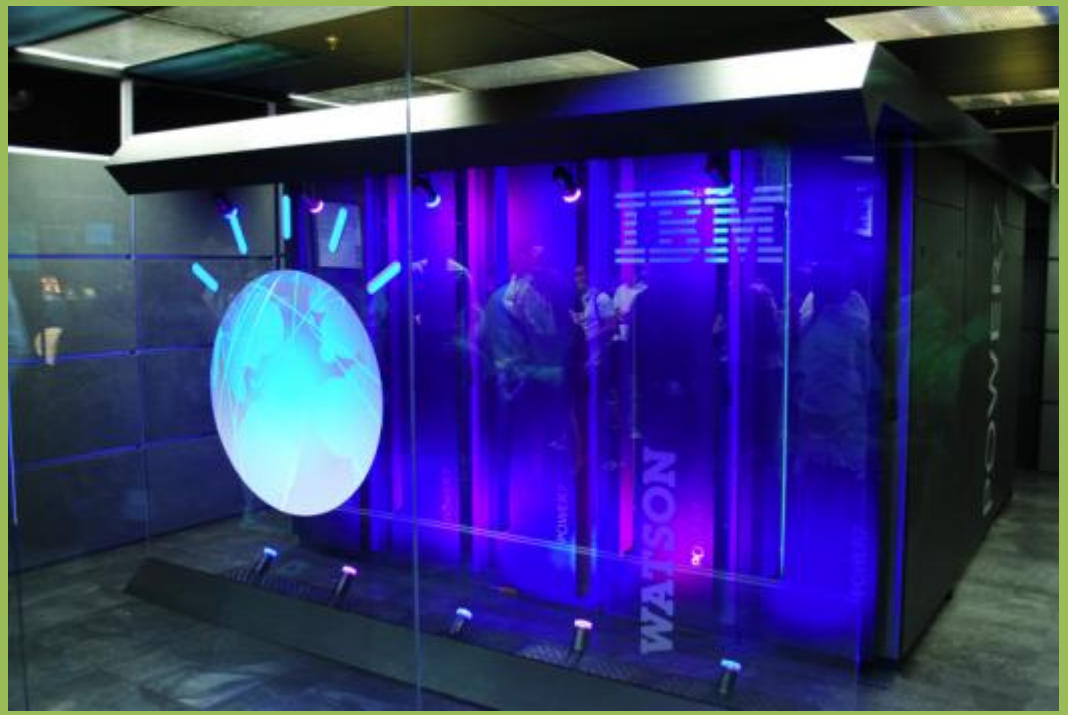
Jest niemal pewne, że konkurenci w mistrzostwach GGP uruchamiają programy na potężnym sprzęcie (CadiaPlayer, Ary, TurboTurtle, Maligne).

Trzeba „jechać tym samym pociągiem”

Zrównoleglenie jest ***prawdopodobnie*** koniecznością, aby nie pozbawić się szansy walki o mistrzostwo.

Po co zrównoleglać?

3. Najmocniejsi gracze komputerowi – w przełomowych pojedynkach i konkursach są uruchamiani w środowiskach zrównoleglonych.



Po co zrównoleglać?

4. Moc obliczeniowa jest coraz tańsza.

ROK	PRZYBLIŻONA CENA 1 GFLOPS (w rzeczywistej maszynie)	CENA 1 GFLOPS /uwzględnienie inflacji
1984	15 000 000$	33 000 000$
1997	30 000$	42 000$
04.2000	1000$	1300$
08.2003	82$	52$
08.2012	0.75$	0.73$

5. Możliwość testowania algorytmów i metod związanych ze zrównoleglaniem obliczeń – dodatkowa wartość twórcza.

Które etapy można zrównoleglać?

Odpowiedź: **wszystkie**.

- Etap selekcji
  (operacje na drzewie, wyznaczanie kolejnej strategii)
- Symulacja wielowątkowa
- Wiele symulacji jednocześnie
- Uaktualnianie wyników (w znikomy sposób)
- Wyznaczanie stanów – zrównoleglenie interpretera

Które etapy warto zrównoleglać?

Odpowiedź:

należy znaleźć najdłużej trwające niezależne operacje, w ramach których nie potrzeba synchronizacji

- trzeba dostosować się do danego problemu

Synchronizacja rozumiana jako:

- potrzeba użycia współdzielonego zasobu
- zależność danych wejściowych pewnej równoległej operacji od wyniku innej operacji

Które etapy warto zrównoleglać?

- Etap selekcji

  (operacje na drzewie, wyznaczanie kolejnej strategii)
- Symulacja wielowątkowa
- **Wiele symulacji jednocześnie**
- Uaktualnianie wyników (w znikomy sposób)
- Wyznaczanie stanów – zrównoleglenie interpretera

Podstawowe metody zrównoleglania MCTS

- Leaf Parallelization
- Tree Parallelization
- Root Parallelization

On the Parallelization of UCT – Cazenave, Jouandeau

W GGP dwie prace związane z graczem Ary:

[1] *Tree Parallelization of Ary on a Cluster*

[2] *A Parallel General Game Player*

Jean Mehat, Tristan Cazenave

Leaf Parallelization

- Zwane również **At-the-leaves Parallelization**
- Tylko jeden wątek ma dostęp do drzewa gry, czyli jego przeszukiwania i rozbudowy

- W momencie, gdy metoda wybierze nowy węzeł do kontynuacji:

  Zamiast pojedynczej losowej symulacji, rozgrywanych jest N symulacji równolegle.

  Wyniki są sumowane/uśredniane.

Tree Parallelization

- Zwane również **Multiple-Runs Parallelization**
- Jeden główny wątek
- Jedno drzewo gry
- Wiele wątków podgraczy (**subplayer**)

W publikacji [1] podgraczem był cały komputer dostępny przez TCP/IP, traktowany jako jeden wątek – taki komputer wykonywał tylko jedną symulację w danym momencie.

Tree Parallelization

Gdy główny wątek wyznaczy stan do kontynuacji:

- pobierany jest kolejny wolny **subplayer**
- przekazywane są mu niezbędne dane: pełny stan gry *(reguły otrzymał na początku)*
- **subplayer** przeprowadza symulację i zwraca wynik
- „Virtual Loss”

W przypadku, gdy każdy **subplayer** jest zajęty, symulację wykonuję główny wątek.

Root Parallelization

- Zwane również **Single-Run Parallelization**
- Wiele równorzędnych wątków wykonujących swoje algorytmy UCT
- Każdy wątek posiada swoje drzewo UCT
- Jeden główny wątek wyróżniony tylko po to, aby komunikować się z GameMasterem

Root Parallelization

- Pod koniec dostępnego czasu, główny wątek zbiera sumaryczne wyniki od reszty wątków
- W oryginalnej, najprostszej wersji następuje uśrednienie wyników (suma / liczba odwiedzin)

- Metoda stosowana w programie do gry w Go: **CrazyStone**.

Podejście MiNI-Player 2012

MiNI-Player 2012

Dwie instancje programu:

Master

[singleton, uruchomiony na jednym komputerze]

Slave

[x3, na pięciu pozostałych komputerach]

- proces Master ma pełną informację o ilości dostępnych procesów Slave i ich adresach
- procesy typu Slave nasłuchują na połączenia od Master

Plik konfiguracyjny

<czy proces jest Master czy Slave>

<lokalny adres do nasłuchiwania połączeń>

<max ilość wątków do wykorzystania>

<zdalny adres #1>

<zdalny adres #2>

....

<zdalny adres #N>

} opcjonalnie

Master process

Jeden główny wątek GUI; *event-driven*

- po upływie odpowiedniego czasu pojawia się zdarzenie

Max N wątków lokalnych (lokalnych robotników)

Max M wątków zdalnych (do obsługi połączeń)

N, M – liczby z pliku konfiguracyjnego

Wątki są typu *long-running thread*.

Raz utworzone istnieją do zakończenia procesu lub gdy logika programu je wyłączy (same się nie kończą).

Master process– wątek GUI

Czekaj na wiadomość START

Po otrzymaniu START:

- Zapisz podstawowe dane (rola, limity czasowe)
- Inicjalizuj puste drzewo gry i inne niezbędne struktury
- Utwórz **N wątków lokalnych** (lokalnych robotników)

  przekaż każdemu reguły gry i nazwę roli

  każdy wątek lokalny tworzy swój **Interpreter Reguł** i **Symulator Gry**
- [PO 1000 milisekundach] uruchom **algorytm skalowania**

Wątek lokalny

```
Inicjalizacja (interpretera, symulatora)
while(true)
{
  If(httpResponseTime)
     EnableProcessing();

  if(node != null)
     DoWork();

  lock(tree)
  {
    if(node != null)
       Update(node, strategy, result);

    PrepareContinuation();
  }
}
```

```
Application.DoEvents();
 Sleep(100);
```

```
result = Simulate(node, strategy);
```

```
tree.Update(node, result);
strategyEvaluator.Update(strategy, result);
```

```
node = tree.ComputeContinuation();
strategy = evaluator.GetNextStrategy();
```

Wątek obsługi zdalnej

```
Inicjalizacja (interpretera, połączenia)
while(true)
{
  If(httpResponseTime)
     EnableProcessing();

  if(node != null)
     DoWork();

  lock(tree)
  {
     if(node != null)
        Update(node, strategy, result);

     PrepareContinuation();
  }
}
```

```
Application.DoEvents();
 Sleep(100);
```

```
requestMsg  = Serialize(node.State);
requestMsg Append(node.Depth, strategy);
SendMessage(requestMsg );

responseMsg = ReceiveMessage();
result = Deserialize(responseMsg);
```

```
tree.Update(node, result);
strategyEvaluator.Update(strategy, result);
```

```
node = tree.ComputeContinuation();
strategy = evaluator.GetNextStrategy();
```

Sekcja krytyczna - UCT

1. Najpierw próbujemy każdy automat raz
2. Następnie według wzoru:

$$a^* = arg \max_{a \in A(s)} \left\{ Q(s,a) + C \sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}$$

a* - akcja do wybrania

s - bieżący stan

Q(s,a) - średni wynik gry przy wykonaniu akcji ***a*** w stanie ***s***

N(s) - liczba dotychczasowych odwiedzin stanu ***s***

N(s,a) - liczba dotychczasowych wyborów akcji ***a*** w stanie ***s***

Algorytm skalowania

W General Game Playing jest jeden główny czynnik wpływający na wydajność zrównoleglania:

Średni czas symulacji Monte-Carlo od stanu początkowego do terminalnego.

Niestety, w przeciwieństwie do programów specjalizowanych do gry w konkretne gry, wartość jest zmienna zależnie od podanych reguł gry.

Algorytm skalowania

Uruchamiany **po 1 sekundzie** od rozpoczęcia symulacji.

Dobrany empirycznie na sprzęcie na konkurs 2012 na podstawie zbioru ponad 20 gier.

W zależności od uzyskanej liczby symulacji:

- Wyłączanych jest $N \geq 0$ wątków lokalnych
- Uruchamianych jest $N \geq 0$ wątków obsługujących połączenia
- Programy zdalne (SlaveProcess) otrzymują parametr **RemoteSimUnit**, który oznacza ile symulacji mają wykonać przed zwróceniem wyniku (zwracany wynik jest **uśrednionym wynikiem** z wykonanych symulacji)

Algorytm skalowania

Uruchamiany po 1 sekundzie od rozpoczęcia symulacji.

Dobrany empirycznie na sprzęcie na konkurs 2012 na podstawie zbioru ponad 20 gier.

Liczba symulacji po 1 sekundzie	Liczba wątków (N)	Liczba połączeń (M)	Liczba symulacji zdalnych (RemoteSimUnit)
> 10000	2	0	-
(6500, 10000]	5	0	-
(4000, 6500]	MAX	33%	4
(250, 4000]	MAX	100%	4
(33, 250]	MAX	100%	2
[0, 33]	MAX	100%	1

Obserwacje

Czas komunikacji sieciowej w połączeniu z częstością dostępu do drzewa wpływają na różną jakość skalowania dla różnych gier.

Procentowe porównanie liczby symulacji*

Liczba komputerów	Connect-4 Suicide	Checkers
1	100% (1.000)	100% (1.000)
2	161% (0.805)	183% (0.915)
3	210% (0.700)	270% (0.900)
4	273% (0.682)	361% (0.902)
5	313% (0.626)	442% (0.884)
6	324% (0.540)	519% (0.865)

Podsumowanie podejścia

Próba rozszerzenia możliwości pojedynczej maszyny.

- budowanie możliwie głębokiego drzewa, aby przyspieszyć zbieżność algorytmu UCT
- tylko jedno drzewo gry w całym systemie rozproszonym
- częsta synchronizacja z drzewem
- dość spory narzut komunikacyjny

Podejście ma swoje wady i zalety.

Podsumowanie podejścia

Zalety:

- Wszystkie symulacje pracują na jedno drzewo, więc szansa, że będzie głębsze niż w innych podejściach **(lepsza zbieżność)**
- Bardzo dobra metoda dla wolnych symulacji, mniej więcej > 150ms
- Skalowanie gwarantuje również prawie-optymalność dla symulacji, gdzie optimum to 1-2 wątki *(na mistrzostwach nie należy się spodziewać tak prostych gier)*

Podsumowanie podejścia

Problemy:

- **Duża liczba wątków na głównym komputerze**
- Duży narzut na komunikację i mało perspektywiczny limit
- Silna zależność od średniego czasu wykonania jednej symulacji w danej grze – **źle się skaluje dla pewnego przedziału szybkości wykonania symulacji**

  [mniej więcej między od 5ms do 100ms]
- Mimo starań, nie do końca wykorzystana moc obliczeniowa komputerów zdalnych

Podejście MiNI-Player 2013

Zmiana koncepcji

Ponieważ:

[A] wydajność obliczeń równoległych w ramach jednej maszyny okazuje się do tej pory zadowalająca

[B] symulacje Monte-Carlo ze strategiami są quasi-losowe

Pomysł, żeby maszyny komunikowały się tylko na początku i na końcu dostępnego czasu.

[A] każda maszyna wykonuje obliczenia niezależnie

[B] przed wykonaniem ruchu następuje agregacja wyników

Model hybrydowy

Tree Parallelization + Root Parallelization

Tree Parallelization

- lokalnie, w obrębie każdej maszyny

Root Parallelization

- globalnie, pomiędzy komputerami

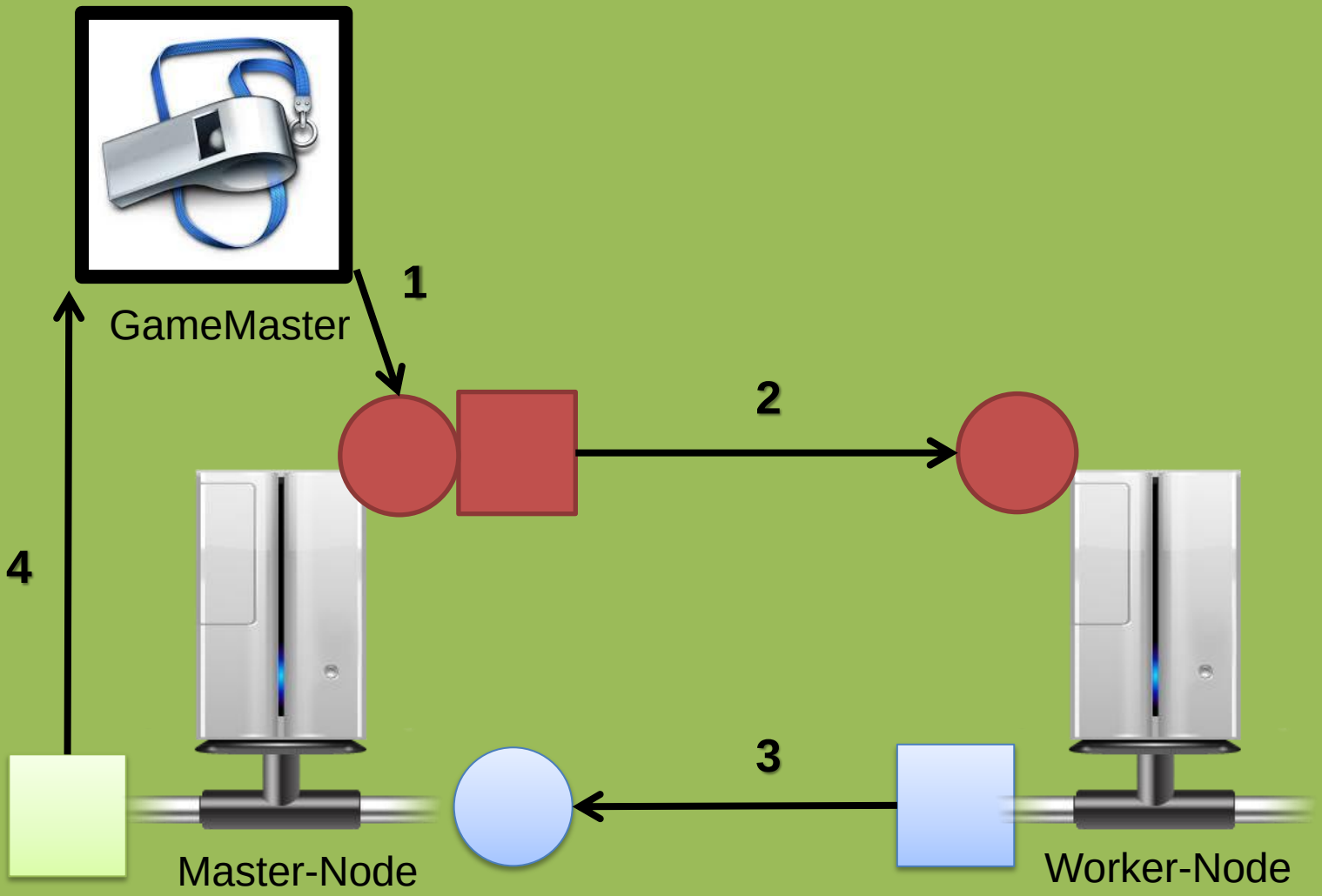
[Root Parallelization – pomysł niezależny, co może dowodzić, że metoda jest intuicyjna]

Zmiany implementacyjne

- Uproszczenie architektury – być może będzie udostępniony kod na zasadzie GNU GPL.
- Użycie Task Parallel Library (TPL) w .NET Framework 4.5
- Wątki tworzone są na czas wykonania zadań i sprzętowo wywłaszczane, gdy skończy się dostępny czas. Zredukowanie ryzyka, że główny wątek nie zdąży odpowiedzieć.
- Wspólny kod - każda instancja programu dziedziczy z jednej klasy *NodeProgram.*

Bardzo mało kodu specjalizowanego dla konkretnego typu węzła.

Komunikacja



Legenda

- Interfejs **przesyłania** wiadomości od GM (START, STOP, PLAY)

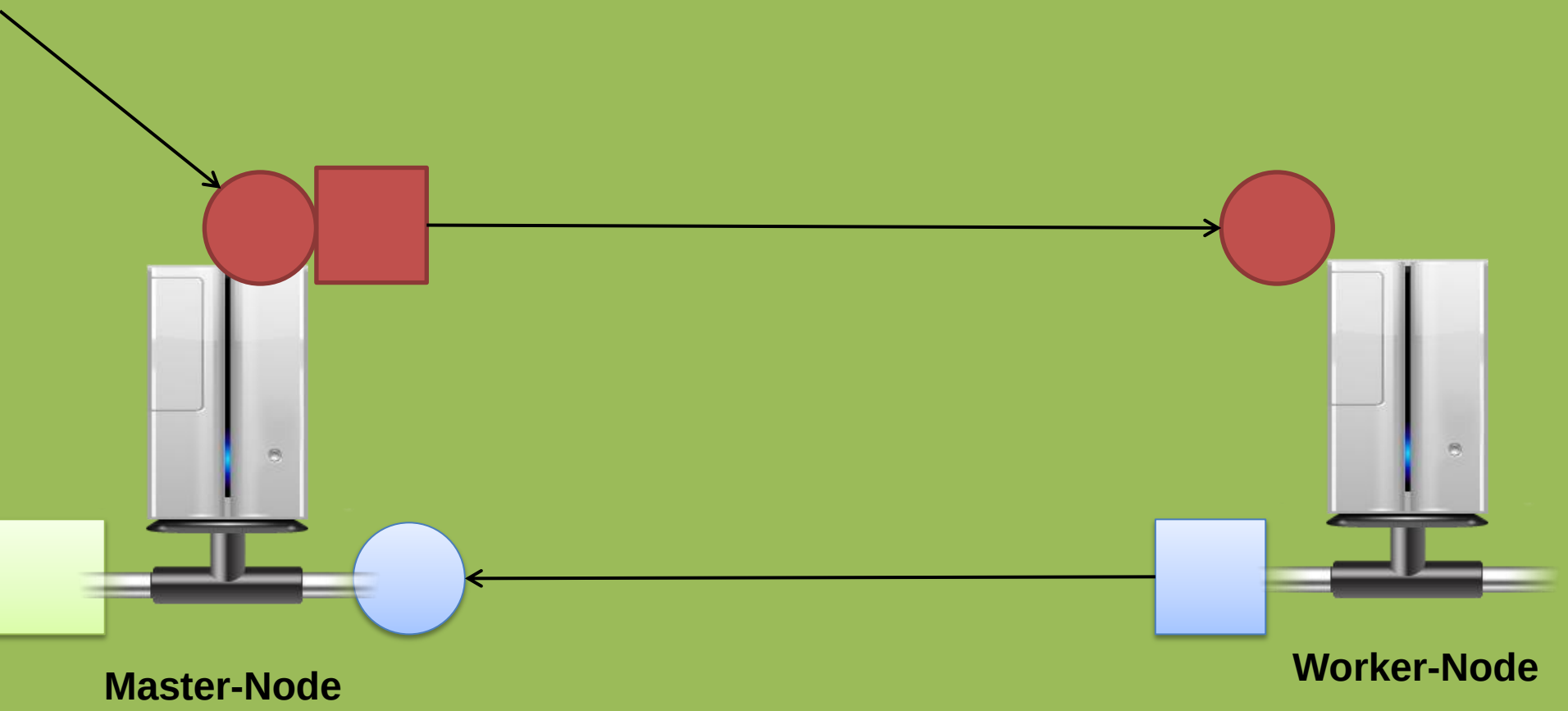
- Interfejs **odbierania** wiadomości od GM (START, STOP, PLAY)

- Interfejs **wysyłania zagregowanych danych symulacji** dla węzłów na poziomie dzieci korzenia (potencjalnych następnych stanów)

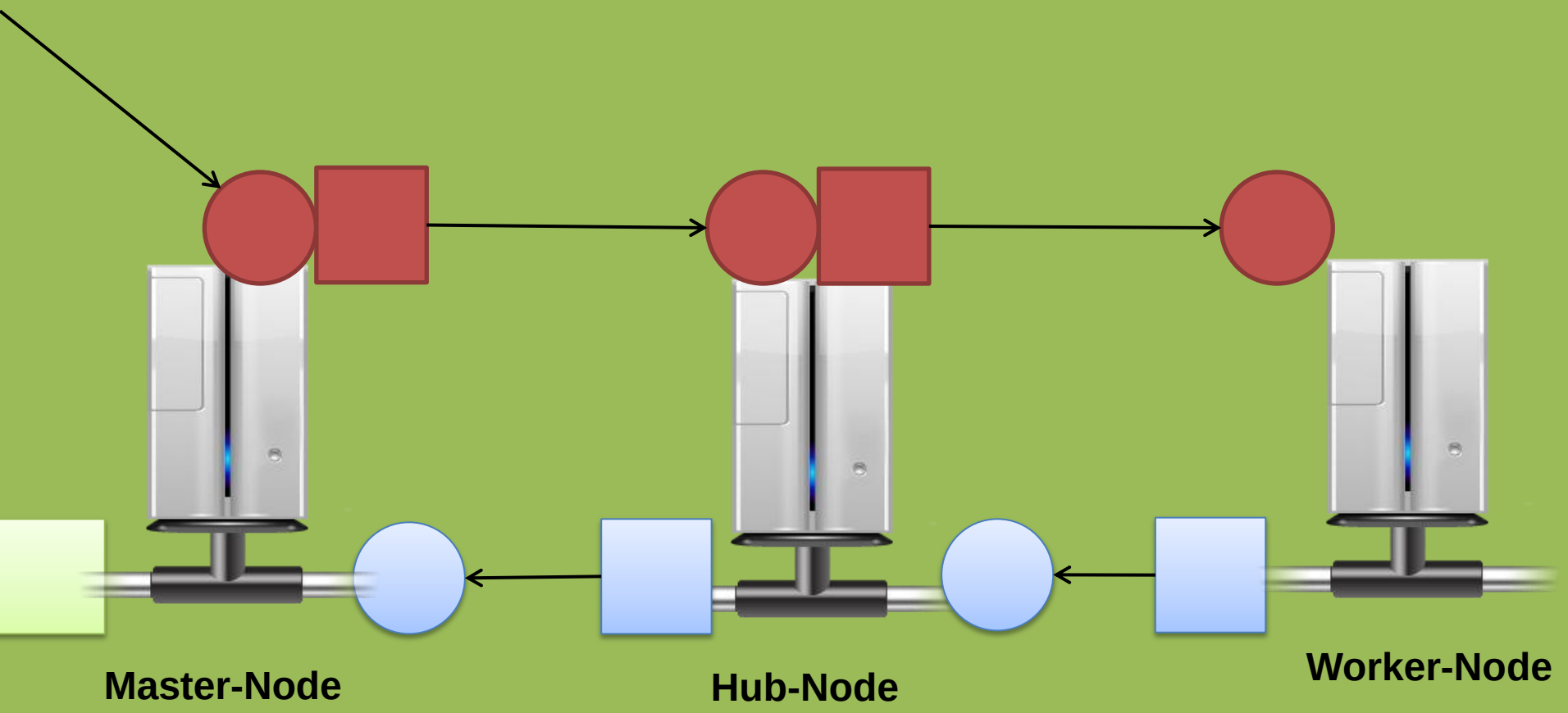
- Interfejs **odbierania zagregowanych danych symulacji** dla węzłów na poziomie dzieci korzenia (potencjalnych następnych stanów)

- Interfejs **wysyłania** odpowiedzi do GM

Master-Workers



Master-Hubs-Workers



Master-Hub-Node

Master-Node

Hub-Nodes

Worker-Nodes

Skalowalność podejścia

Dwie kwestie, które można rozróżnić:

1. Wykorzystanie mocy obliczeniowej i.e. wzrost całkowitej liczby symulacji wraz z dokładaniem komputerów
2. Zależność poprawy jakości gry od dodawania kolejnych komputerów

Skalowalność podejścia

[1] Wykorzystanie mocy obliczeniowej

- Czynnikiem hamującym jest jedynie współczynnik opóźnienia wymagany do komunikacji pomiędzy warstwami (aby nie zgubić wyników).
- Metoda pozwala na zbudowanie platformy bardzo wydajnie wykorzystującej moc

- Ustawienie liczby poszczególnych węzłów, połączeń oraz opóźnień: to już kwestia **dopasowania do konkretnego** sprzętu (klastra, serwerów, sali laboratoryjnej)

Skalowalność podejścia

[2] Skalowanie siły gry

W teorii:

- Jakość gry powinna rosnąć wraz z dodawaniem maszyn
- Punkt nasycenia powinien być daleko, ze względu na znikomy narzut komunikacyjny wraz z dodawaniem kolejnych komputerów

W praktyce:

- Efekt skalowalności zależy od gry
- **Pokażą testy**

Skalowalność liczby symulacji - rozważania

Załóżmy, że mamy dowolnie dużą liczbę komputerów.

Załóżmy, że każdy węzeł **Master/Worker** może wydajnie obsłużyć:

- 4 lokalne wątki robotników

Załóżmy, że każdy komputer **Master/Hub** może wydajnie obsłużyć:

- 12 połączeń od komputerów zdalnych

Pytanie:

Ile maksymalnie komputerów możemy wykorzystać przy:

a) Braku warstwy pośredniej (HubNode)

b) Jednej warstwie pośredniej (HubNode)

c) Dwóch warstwach pośrednich (HubNode)

Ile w każdym przypadku maksymalnie działa wątków robotników?

Skalowalność liczby symulacji - rozważania

Odpowiedź:

Warstw pośrednich	Maksymalna liczba komputerów	Sumaryczna liczba dostępnych wątków
0	13	52
1	157	580
2	1885	6912

Dla jednej warstwy i innych parametrów:

Parametr sprzętu	Maksymalna liczba komputerów	Sumaryczna liczba dostępnych wątków
3 | 8	73	195
12 | 24	601	6924

Otwarte kwestie 1/2

W jaki sposób agregować wyniki uzyskane z komputerów zdalnych?

- proste uśrednianie
- uśrednianie zgodnie z ustaloną wagą np. ważności węzłów
- głosowanie *(majority voting)*
- inne metody rankingowe lub z teorii podejmowania decyzji

Otwarte kwestie 2/2

Co agregować?

- średni wynik ***Q*** i liczbę symulacji ***V*** na poziomie węzłów dzieci-korzenia
- współczynnik ***EV*** *(i w jaki sposób)*
- dane z niższych poziomów w drzewie?
- …. ?

Zmodyfikowana formuła wyboru ruchu (EV)

- MiNI-Player korzysta ze zmodyfikowanej formuły **EQ**, która wykorzystuje dane z 2 poziomów w drzewie

- Wybrany zostaje potomek **Root.Child[i]** o największej wartości **EQ**

Zmodyfikowana formuła wyboru ruchu (EV)

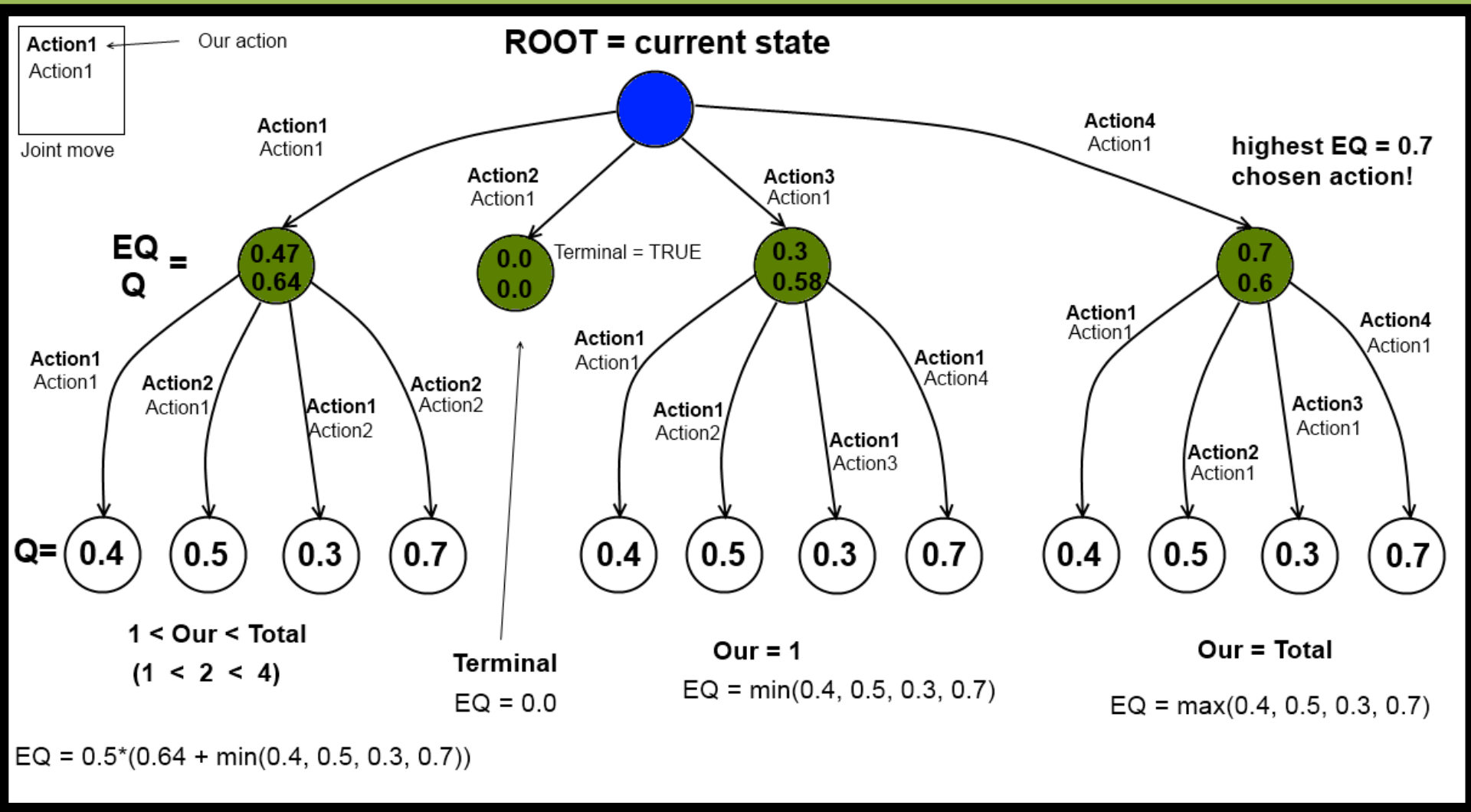
Our **–** liczba unikalnych akcji naszego gracza

Total **–** liczba wszystkich dostępnych akcji łączonych (joint moves)

Iterujemy po węzłach wychodzących z korzenia:

Warunek w węźle	**EQ**
Stan terminalny = TRUE	$\mathrm{EQ} = node.Q \qquad (1.01 * node.Q)$
Our = 1	$\mathrm{EQ} = \min_{i=1\ldots N}(node.Child[i].Q)$
1 < Our < Total	$\mathrm{EQ} = \frac{\left(\min_{\mathrm{i=1\ldots N}}(\mathrm{node.Child[i].Q}) + \mathrm{node.Q}\right)}{2}$
Our = Total	$\mathrm{EQ} = \max_{i=1\ldots N}(node.Child[i].Q)$

Zmodyfikowana formuła wyboru ruchu (EV)



Uwzględnanie EV – sprawa nierozwiązana

Czy obliczać **EV** dla każdego wierzchołka przed przesłaniem?

Potencjalne możliwości agregacji:

- rozszerzenie wzorów, aby stosować **min** i **max** po wszystkich węzłach
- uśrednianie obliczonych **EV** na każdej maszynie
- uśrednianie ważone liczbą symulacji
- wybranie **EV** z najbardziej reprezentatywnych – najlepiej przesymulowanych gałęzi

Planowany efekt

Tree Parallelization + Root Parallelization

Pozytywne:

- Drastyczne zmniejszenie narzutu komunikacyjnego
- Wykonywanie sumarycznie większej ilości symulacji
- Uśrednianie wyników przy pomocy większej próbki
- Możliwość wykorzystania dużo większej liczby komputerów

Negatywne:

- Brak zwiększenia głębokości drzewa gry
- Ryzyko powtarzania symulacji na różnych maszynach

Negatywne efekty

[1] Brak zwiększenia głębokości drzewa gry

- Lokalnie na maszynach jest Tree Parallelization (TP), więc hybryda jest i tak kompromisem w porównaniu do pełnego RP
- Zamiast większej głębokości drzewa mamy pewniejsze wyniki. Nadaje się lepiej do gier o dużej średniej liczbie kroków i dużym BF.

  Metoda tuningowana pod gry **drugiego dnia mistrzostw**.

[Problem pojawi się, jeśli przeszukanie drzewa głębiej pokaże wykaże w pewnym momencie słabość ruchu, który zapoczątkował ścieżkę – punkt przełomowy UCT]

Negatywne efekty

[2] Ryzyko powtarzania symulacji na różnych maszynach

Wiele sposobów na umieszczenie różnych ziaren na komputerach:

- różne parametry strategii (prawdopodobieństwa użycia)
- różne zestawy dostępnych strategii
- różne parametry UCT
- zmienna predefiniowana kolejność wybierania akcji po raz pierwszy do symulacji

[ten aspekt jest na razie na etapie przemyślenia]

Dyskusja: zalety vs wady

Wybór metody niesienie ze sobą pewien element hazardu – która okaże się lepsza dla gier na mistrzostwach?

Większe drzewo zbudowane przy pomocy mniejszej liczby symulacji

VS

Możliwość użycia nawet kilkadziesiąt razy większej liczby maszyn – więcej wykonanych symulacji.

Skłaniam się ku drugiej metodzie.

dziękuję za uwagę